MÀSTER UNIVERSITARI EN ENGINYERIA DE SISTEMES AUTOMÀTICS I ELECTRÒNICA INDUSTRIAL

-Trabajo Final de Máster-

Estudio para el diseño y programación de un algoritmo de planificación de trayectorias para un coche autónomo

Proyectista: Dídac Castellet Menchón

Director: Bernardo Morcego Seix

Convocatoria: Cuatrimestre Primavera 2017/2018

RESUMEN

En una época donde la robótica crece a pasos agigantados, el nuevo reto de los ingenieros se está convirtiendo en desarrollar robots móviles completamente autónomos. Por ello, las técnicas de inteligencia artificial están en continuo desarrollo.

Este trabajo se centra en el estudio y diseño de los algoritmos de planificación de trayectorias para un coche autónomo.

En la primera parte del trabajo, se han estudiado los diferentes algoritmos de planificación de movimiento de manera independiente. Se han observado sus diferentes cualidades, ventajas y desventajas. Una vez se ha hecho una pre-elección de los algoritmos que se utilizarán, se han diseñado y programado para poder compararlos entre sí y finalmente se ha escogido uno definitivo.

En la segunda parte del trabajo se ha adaptado el planificador escogido a las características del coche mediante modificaciones necesarias para que el vehículo pueda ser controlado. Seguidamente se ha diseñado un controlador borroso para el coche autónomo.

Para finalizar, una vez se ha diseñado el control del vehículo, éste se ha implementado en el planificador. A continuación se han realizado varias pruebas con circuitos diferentes y se ha estudiado el comportamiento del algoritmo de planificación de trayectorias.

ABSTRACT

In a time where robotics is growing exponentially, the new challenge for engineers is developing into completely autonomous mobile robots. That is why artificial intelligence techniques are in a continuous developing.

This work focuses on the study and design of trajectory planning algorithms for an autonomous car.

In the first part of the work, the different motion planning algorithms have been studied in an independently way. Their different qualities, advantages and disadvantages have been observed. Once a pre-election of the algorithms that will be used has been made, they have been designed and programmed in order to compare them with each other and finally, a definitive one has been chosen.

In the second part of the work, the chosen planner has been adapted to the characteristics of the car through necessary modifications so that the vehicle can be controlled. Then, a fuzzy controller for the autonomous car has been designed.

Finally, once the control of the vehicle has been designed, it has been implemented in the planner. Subsequently, several tests have been carried out with different circuits and the behaviour of the trajectory planning algorithm has been studied.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABLAS

MEMORIA

1 - INTRODUCCIÓN

1.1 - Robótica y planificadores de movimiento

La robótica y todo lo que ésta engloba es una de las disciplinas que más ha crecido, y está creciendo, durante los últimos años. Cada vez con más frecuencia las máquinas o robots sustituyen a las personas que realizan trabajos repetitivos, mecánicos y monótonos. Sin embargo, muchas de estas máquinas necesitan la supervisión constante de un técnico que la guíe y la configure adecuadamente.

El objetivo de los científicos ha ido variando y ahora en vez de focalizarse en diseñar las máquinas más potentes se centran en la autogestión de estas, que puedan tomar sus propias decisiones. El principal objetivo actual se centra en la **inteligencia artificial**, y en su posible implementación en cualquier tipo de máquina o robot, la cual permita ayudar a las personas en cualquier ámbito de la vida y no sólo en el mundo de la industria.

En lo que respecta a la robótica móvil, uno de los retos consiste en caracterizar el entorno por el cual los robots deben desplazarse, identificar obstáculos, zonas de paso e incluso ubicarse con la mayor precisión posible con respecto a un sistema de referencia dado. Una vez hecho esto, el robot debe trazar una trayectoria hacia su destino, y después tratar de seguirla con el mínimo error posible. A veces, no se dispone de ningún mapa y el robot ha de maniobrar de acuerdo a lo que detecta a través de sus sensores. Este tipo de técnicas reciben el nombre de **navegación reactiva** y una de las formas de implementarla es generando trayectorias destinadas a evitar los obstáculos que el robot va reconociendo. El cálculo de trayectorias en un entorno dado es un aspecto fundamental en el ámbito de la robótica móvil.

De forma genérica, el problema de la planificación de movimientos en robots móviles consiste en determinar una trayectoria admisible o conjunto de movimientos que permitan llevarlos desde una configuración inicial hasta otra final dentro del espacio de configuraciones libres de colisión. Se entiende por trayectoria admisible aquella que se reproduce siempre dentro del espacio libre de colisión y al mismo tiempo cumple con las restricciones cinemáticas del vehículo.

Este problema ha sido ampliamente estudiado en la literatura de los años 90 (Aurenhammer, 1991; Latombe, 1991; Laumont y otros, 1994; Muñoz, 1995; LaValle, 1998), describiendo múltiples formas de solventarlo.

Entre éstas soluciones se hallan los denominados **métodos *"roadmap"***, caracterizados por construir una descripción del espacio libre con la forma de una red de curvas bidimensionales.

Los métodos *roadmap* construyen un grafo que representa en sus nodos algunas configuraciones factibles del robot, y en sus arcos la posibilidad constatada de conexión entre dos nodos. Una vez construido este grafo, para hallar un camino entre el origen y el destino simplemente se deben conectar dichos puntos con el grafo y hallar una secuencia de nodos dentro de él.

Existen varios métodos *roadmap* como el diagrama de Voronoi, descomposición en celdas, grafos de visibilidad, campos de potencial, etc. Más adelante se explicará el funcionamiento de alguno de ellos aunque brevemente, ya que el estudio de cada método no es el principal objetivo de este trabajo.

Las limitaciones de estos métodos pueden ser excedidas debido a la posible complejidad del entorno, el número de grados de libertad y a las restricciones cinemáticas y dinámicas que pueda presentar el robot móvil. Además, para hallar una posible solución el tiempo de cómputo suele incrementarse excesivamente. Frente a este problema, los **métodos estocásticos** ofrecen interesantes alternativas por su sencillez de implementación y su comportamiento ventajoso en algunos tipos de escenarios.

Dentro de los métodos aleatorios se pueden encontrar los mapas probabilísticos o ***"Probabilistic Roadmaps"* (PRMs)** (Amato y Yan, 1996; Kavraki y otros, 1996a) y el algoritmo ***"Rapidly Exploring Random Trees"***, o RRT (LaValle, 1998). En apartados posteriores se explicará el funcionamiento de estos dos métodos, centrándose en el método RRT. Éste algoritmo se ideó en principio como herramienta de apoyo para ser integrado en otros planificadores. Sin embargo, utilizado como planificador autónomo reveló cualidades muy interesantes. Por este motivo hay una gran abundancia de artículos y publicaciones hablando del RRT y de posibles mejoras e implementaciones.

1.2 - Problema a resolver

El problema en cuestión consiste en el estudio y creación de un algoritmo de planificación de movimiento capaz de ser implementado en un robot móvil, más concretamente en un coche autónomo.

Una vez se haya creado el algoritmo con todos los parámetros necesarios, se le implementará el control cinemático del coche de manera que siga la ruta especificada por el planificador.

Para que el planificador pueda funcionar, previamente debe recibir el punto inicial del coche, el punto final y ciertos puntos de paso. En la figura 1.1 se muestra un ejemplo de la información que el algoritmo debe conocer.

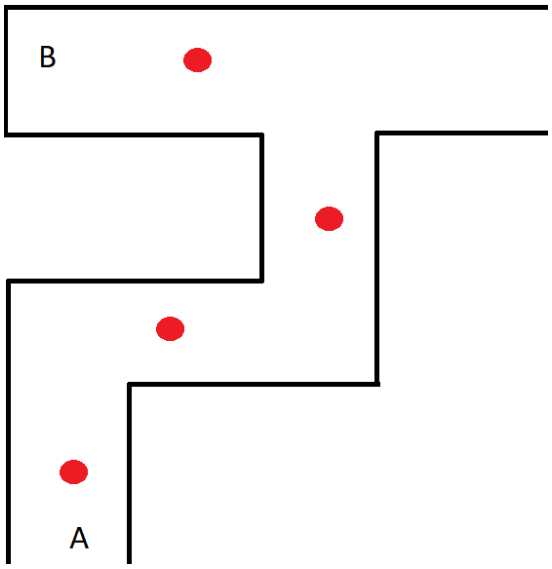

Figura 1-1. Esquema de la información necesaria para el funcionamiento del planificador.

A es la posición inicial, los círculos rojos son los puntos por donde el coche debe pasar y B la posición final.

Con estas variables, el planificador deberá trazar una trayectoria que no colisione con ningún obstáculo y además deberá cumplir con las restricciones cinemáticas del vehículo.

1.3 - Objetivos

1.3.1 - Objetivo principal

El principal objetivo de este trabajo es el **diseño y programación de un algoritmo de planificación de movimiento**, el cual pueda ser integrado en el control de un coche autónomo o un vehículo móvil.

El objetivo se cumplirá siempre y cuando el vehículo siga la ruta planificada por el algoritmo y no haya ninguna colisión.

1.3.2 - Objetivos específicos

Una vez se tiene claro el objetivo principal del proyecto, a continuación se detallan los objetivos secundarios pero necesarios para la consecución de este:

- Diseño y programación de un algoritmo RRT el cual se pueda parametrizar.
- Adaptación del algoritmo inicial para que pueda ser leído por el vehículo.
- Diseño del control cinemático del coche.
- Integración del RRT en el control del coche. Se harán las adaptaciones necesarias para la correcta implementación de los dos programas.
- Prueba del algoritmo. Se observará el comportamiento del vehículo con diferentes parámetros y en diferentes circuitos.

1.4 - Alcance del proyecto

Las siguientes tareas y documentos se considerarán parte del proyecto:

- Estudio de los algoritmos de planificación, profundizando en los algoritmos RRT.
- Elección de un algoritmo RRT y su posterior diseño.
- Pruebas con el algoritmo programado.
- Programación del control cinemático del coche.
- Integración del algoritmo y el controlador en un mismo programa.
- Simulación del comportamiento del vehículo en Matlab.
- Memoria del proyecto.
- Anexos

Los siguientes puntos se consideran fuera del alcance del proyecto:

- Estudio de un sistema de control avanzado para el vehículo.
- Cálculo de los puntos de referencia (puntos por dónde el vehículo tiene que pasar). Estos puntos son calculados con anterioridad a la actuación del algoritmo RRT.
- Implementación y simulación de interrupciones (coches, personas, semáforos, etc.)
- Implementación del algoritmo en un vehículo real.

2 - ALGORITMOS DE PLANIFICACIÓN DE MOVIMIENTO

En este capítulo se plantea el problema de planificación de movimiento en el campo de la robótica móvil. Se describen y comparan varios algoritmos y se hace hincapié en los algoritmos RRT.

2.1 - Introducción

La robótica aborda la automatización de sistemas mecánicos que tienen capacidades de detección, actuación y computación. Una necesidad fundamental en robótica es la de tener algoritmos que conviertan especificaciones de alto nivel de tareas a realizar, hasta descripciones de bajo nivel de cómo moverse. Los términos **planificación de movimiento (*Motion Planning*)** y **planificación de trayectoria (*Trajectory Planning*)** se utilizan a menudo para estos tipos de problemas.

La planificación del movimiento generalmente ignora la dinámica y otras limitaciones diferenciales y se centra principalmente en las traslaciones y rotaciones requeridas para mover el objeto. También suele tener en cuenta otros aspectos como posibles incertidumbres, errores de modelado y optimalidad. Por el contrario, la planificación de trayectorias se refiere, generalmente, al problema de determinar el movimiento del robot de una manera que respete sus limitaciones a lo largo de una solución de planificación de movimiento previamente establecida. Esta diferenciación de los términos "*Motion Planning"* y "*Trajectory Planning"* está recogida en el libro *Planning Algorithms,* [5] de *Steven M. LaValle*.

Una versión clásica de la planificación de movimiento se denomina a veces "Problema del Movimiento del Piano". Para este problema, se tiene en cuenta que hay un diseño preciso asistido por computadora (CAD) de un modelo de una casa y un piano como entrada a un algoritmo. Este algoritmo deberá determinar cómo mover el piano de una habitación a otra de la casa sin golpear ningún obstáculo.

En **inteligencia artificial**, los términos planificación y planificación de IA adquieren un significado más discreto. En lugar de mover un piano a través de un espacio continuo, como en el problema de planificación de movimiento del robot, la tarea consistirá en resolver un rompecabezas de manera discreta. Aunque tales problemas podrían

modelarse con espacios continuos, la discretización de los espacios por donde no podrá circular el robot, por ejemplo, hará más fácil encontrar una solución al problema.

A continuación, se describen algunos conceptos básicos sobre algoritmos de planificación de movimiento:

- **Espacio de configuraciones:** Conjunto de todas las configuraciones que puede adoptar el robot independientemente del escenario en el que se ubique. La dimensión de este espacio será igual al número de parámetros necesario para describir una configuración. Habitualmente se denota como *C*. En este caso, su representación consiste en dos variables coincidentes con la posición cartesiana de su centro (o de otro punto de referencia del mismo) y un ángulo que define su orientación. Si el vehículo sólo se mueve en el plano, se hablaría de un espacio de dimensión 3.

- **Colisión:** Se dice que un robot entra en colisión cuando su configuración sitúa uno o varios de sus elementos intersecando el espacio ocupado por obstáculos o los límites del escenario de estudio.

- **Obstáculo:** Para el ámbito de la planificación, un obstáculo se considera como un conjunto cerrado y acotado de puntos del espacio físico. Del obstáculo sólo interesa el volumen y la posición que ocupa. Así, un obstáculo también puede ser un lugar que por alguna razón no se desea que el robot ocupe.

- **Obstáculo-C:** Sea *q* un elemento de *C*. Se denota como *A(q)* el espacio físico que ocupa el robot cuando adopta la configuración *q*. Sea B_i un obstáculo, se define el obstáculo-C asociado a B_i (abreviadamente CB_i), como el conjunto de configuraciones cuyo espacio físico colisiona con B_i. Es decir, CB_i es el efecto de B_i en el espacio de configuraciones:

$$CB_i = \{ q \in C : A(q) \cap Bi \neq 0 \} \qquad (2.1)$$

  La existencia de colisión del robot con un obstáculo implica que el espacio que ocupa, *A(q)*, es compartido con el que ocupa el obstáculo, B_i, o lo que es lo mismo, la intersección de *A(q)* y B_i no es nula porque existen puntos de colisión.

- **Región de obstáculos-C:** Es la unión de todos los obstáculos C. Se denota como:

$$CB: n = número\ de\ obstáculos$$

- **Espacio de configuraciones libres de colisión:** Es el subconjunto del espacio de configuraciones cuyos elementos no presentan colisión. Habitualmente se denota como C_{free}.

$$C_{free} = C \setminus CB \qquad (2.2)$$

2.2 - Métodos *roadmap*

Como se ha explicado en la introducción, estos métodos son bastante visuales, ya que se basan en la construcción de un grafo, dividido normalmente en nodos y arcos. Sin embargo, debido a sus limitaciones, la complejidad del entorno y las restricciones del robot pueden suponer un problema para estos algoritmos.

2.2.1 - Grafos de visibilidad

Éste método de planificación se basa en la construcción de un grafo, en el cual sus nodos representan configuraciones factibles del robot mientras que sus arcos representan el coste de conexión directa entre dos vértices. Una vez construido dicho grafo, denominado grafo de conectividad, hallar un camino entre el origen y el destino se reduce a conectar dichos puntos con el grafo y hallar una secuencia de nodos dentro de él [1]. El término de visibilidad alude a que en este método se enlazan los vértices entre los que existe visión directa, es decir, sin obstáculos que lo impidan.

El método de grafos de visibilidad, como otros planificadores, precisa de una descripción poligonal de los obstáculos-C. Así, el entorno de estudio queda definido por los límites del plano en el que se desplaza el robot y un conjunto de polígonos que identifican los obstáculos. Dentro del espacio de configuraciones libres de colisión se sitúan las posiciones de partida y meta. De esta manera el problema queda totalmente definido.

El proceso comienza generando una lista de los vértices de los polígonos a los que se añaden la posición inicial y final. Con estos puntos se genera una matriz cuadrada con tantas filas como puntos. A continuación, se evalúa para cada par de puntos su posibilidad de enlace directo, es decir, si el robot ubicado en uno de los puntos puede alcanzar otro punto desplazándose en línea recta. Si un obstáculo no lo impide, esto

será posible y la longitud del tramo a recorrer se almacenará en la matriz, indicando el coste de realizar dicho enlace. Si por el contrario existe obstáculo, en la matriz se anotará un valor que indique la imposibilidad de enlace directo. Esta matriz no es más que la descripción de un grafo que une los puntos entre los que es posible el enlace directo. En la figura 2-4 se observa un ejemplo de este método. Una vez se han generado todos los vértices y todas las rectas teniendo en cuenta los obstáculos-C, se puede encontrar una solución.

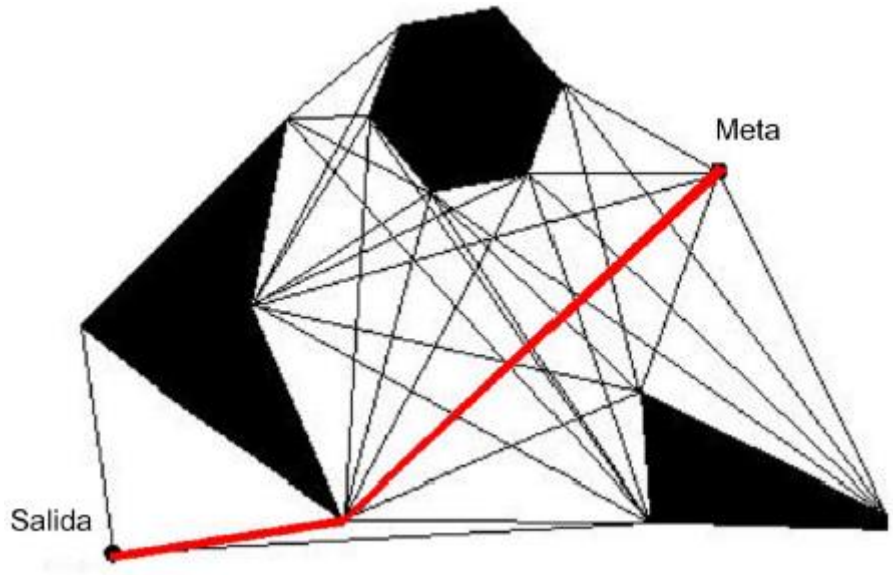

Figura 2-1. Grafo de Visibilidad.

Recorriendo este grafo se puede alcanzar cualquier vértice. Así pues, una solución, se puede conseguir escogiendo una sucesión de vértices conexos dentro del grafo en la que el primero sea el punto de partida y el último la meta. Entre los muchos caminos a escoger se suele utilizar el criterio de mínima distancia. Para ello, se suele utilizar el algoritmo Dijkstra de caminos mínimos, aunque hay otros que también se pueden utilizar.

El coste computacional de este algoritmo se eleva ostensiblemente con el número de vértices. Sin embargo, existen ciertas mejoras como por ejemplo la eliminación de vértices cóncavos en el cálculo del grafo. Ello se debe a que la trayectoria que utilice un vértice cóncavo será siempre de mayor longitud que otra que pase por los mismos puntos y sustituya dicho vértice por un punto próximo de su bisectriz. Esto demuestra que no es la trayectoria óptima y por tanto, todos los arcos del grafo pertenecientes a dicho vértice jamás estarán en la solución.

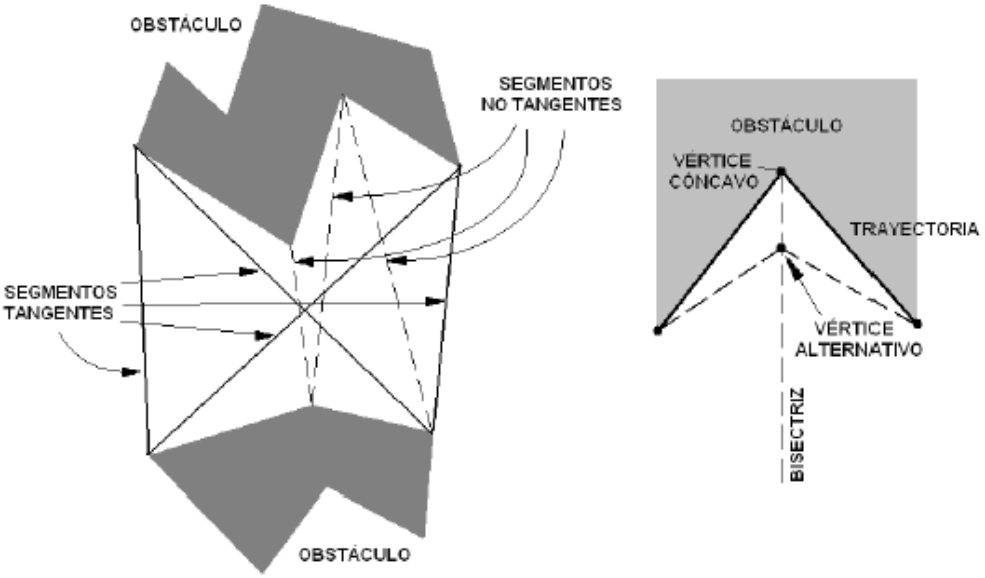

Figura 2-2. Segmento tangente y vértice cóncavo.

Otro concepto interesante que aporta una reducción significativa en el grafo, es el de segmento tangente a un polígono. Sean dos polígonos *A* y *B*, *X* e *Y* dos vértices de los mismos ($X \in A$, $Y \in B$), *L* la línea que pasa por *X* e *Y*. Se dice que el segmento *XY* es tangente si *L* no intersecta *A* ni *B*. Se demuestra que la solución dentro del grafo de visibilidad, si existe, está compuesta por segmentos tangentes. Así se eliminan bastantes arcos del grafo, disminuyendo el coste computacional.

Existen ciertas peculiaridades asociadas a la aplicación práctica de este algoritmo. Tal y como se describe en la introducción, un obstáculo, aunque tenga la morfología de un polígono, su proyección en el plano no tiene por qué coincidir con el obstáculo. Para simplificar esta cuestión (facilitar la implementación informática), se aumenta el área del polígono de modo que inscriba el espacio dentro del cual el robot, al menos en alguna orientación, pueda colisionar. Es habitual tomar un punto de referencia para el robot, de modo que la descripción de su posición (su configuración) se establezca con las coordenadas cartesianas de este punto de referencia y su orientación (ángulo del eje central con respecto al eje de abscisas).

Por último, si se desea utilizar directamente la solución aportada por este algoritmo, hay que tener en cuenta un aspecto relevante y es que el robot debe ser capaz de girar

sobre sí mismo. Si se observa la solución del algoritmo de visibilidad, ésta consiste en una secuencia de segmentos unidos por vértices. El robot, para seguirla, debería avanzar recto por cada segmento y orientarse en cada vértice sin abandonarlo. Esto es posible en algunos robots como los vehículos de conducción diferencial, sin embargo existen otros en los que resulta del todo imposible por su configuración.

2.2.2 - Diagramas de Voronoi

Los Diagramas de Voronoi son uno de los métodos de interpolación más simples, basados en la distancia euclidiana, especialmente apropiada cuando los datos son cualitativos. Los Diagramas de Voronoi son en realidad los grafos que dan el nombre al método. En este caso el grafo resultante tiene la ventaja de que maximiza la distancia entre el robot y los obstáculos. Al igual que en el anterior método, la aplicación de éste a la planificación exige el modelado de los obstáculos como obstáculos-C poligonales

Un diagrama de Voronoi generalizado (denotación utilizada para diferenciarlo del original, en el que los obstáculos se reducían a puntos) representa los puntos que equidistan de elementos vecinos entre sí. Estos elementos pueden ser obstáculos o bien bordes del escenario. El atributo de vecindad expresa que no hay ningún otro elemento entre dos que sean vecinos.

Dado que el lugar geométrico de los puntos del plano que equidistan de un punto y una recta es una parábola, y el lugar geométrico de los puntos del plano que equidistan de dos rectas es a su vez una recta, si los obstáculos son polígonos, el diagrama estará compuesto de una sucesión de parábolas y segmentos rectos interconectados. Si se toman los puntos de interconexión como vértices, y los fragmentos de parábolas y rectas como arcos, se dispondrá de un grafo. No obstante, los diagramas de Voronoi también pueden extraerse de escenarios donde existan obstáculos con lados curvos.

Una vez generado el grafo, se procede a conectar el punto de partida con la meta. Para encontrar una solución hay que tener en cuenta la disposición del punto inicial y final.

La disposición de los puntos se puede clasificar de la siguiente manera:

- Entre dos segmentos
- Entre un vértice y otro elemento

Si el punto se halla en la vecindad de dos segmentos, se encuentra la línea recta ortogonal que une dicho punto al segmento más próximo. El segmento de dicha recta definido por su intersección con el grafo y el punto considerado, constituirá la unión de este punto al grafo.

Si por el contrario el punto se halla en la vecindad de un vértice y otro elemento, se encuentra la recta que pase por el vértice y el punto, determinándose de la misma forma que en el caso anterior el arco del grafo necesario. Un ejemplo de este caso se puede ver en la figura 2-6.

A partir de aquí el problema se resuelve como en el caso anterior, es decir, se aplica el algoritmo de Dijkstra, por ejemplo, y se halla una secuencia de vértices en el grafo que una sendos puntos.

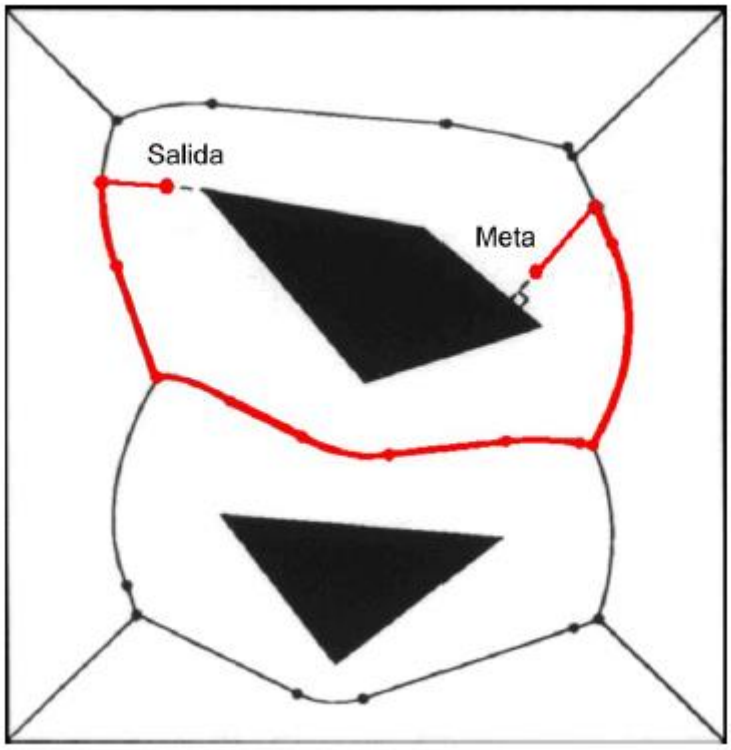

Figura 2-3. Diagrama de Voronoi con los puntos situados entre vértice y segmento

Al igual que en los grafos de visibilidad, si se desea utilizar el grafo como camino para ser seguido por un robot, será necesario asumir la capacidad de maniobra del mismo para girar abruptamente en algunos vértices. Por otro lado, la ventaja de este método

radica en su capacidad para alejarse de los obstáculos, eligiendo siempre el camino más despejado posible. Esto puede ser muy interesante en escenarios con abundantes obstáculos y pasos estrechos, sin embargo resulta un inconveniente en entornos con escasos obstáculos. Dependiendo de la ubicación de los mismos, el diagrama podría arrojar trayectorias que diesen rodeos largos e innecesarios para llegar a la meta.

2.3 - Métodos de generación aleatoria

Los métodos de generación aleatoria suponen una ventaja frente a los métodos *"roadmap"* por su sencilla implementación y su capacidad de adaptación a ciertos escenarios. A continuación se presentan dos de los métodos más importantes.

2.3.1 - Mapas probabilísticos (PRMs)

Otro de los algoritmos de planificación de actualidad se conoce como *"Probabilistic roadmap*", mapas probabilísticos o PRMs. Una de sus principales virtudes consiste en su eficacia en el cálculo de trayectorias de robots con muchos grados de libertad. Existen dos tipos fundamentales de mapas probabilísticos: Los **PRM de consulta única** y los de **múltiple consulta**.

La idea básica de los PRM es tomar muestras aleatorias del espacio de configuración del robot, probándolas para ver si están en el espacio libre, y usar un planificador local para intentar conectar estas configuraciones a otras configuraciones cercanas. Las configuraciones de inicio y de objetivo se agregan, y se aplica un algoritmo de búsqueda de gráficos al gráfico resultante para determinar una ruta entre las configuraciones de inicio y objetivo.

Los PRM de consulta única mantienen un único grafo y restringen la búsqueda a aquellas configuraciones que sean accesibles desde los nodos de este grafo. También pueden construirse dos árboles simultáneos, cada uno de los cuales tendrá como raíz a una de las configuraciones que se desean unir con una trayectoria, y cuyo crecimiento se detendrá en el momento en que sea posible establecer una conexión entre los dos árboles. En el primer caso (árbol único) se dice que se hace una búsqueda unidireccional, mientras que en el segundo caso (dos árboles) se trata de una búsqueda bidireccional.

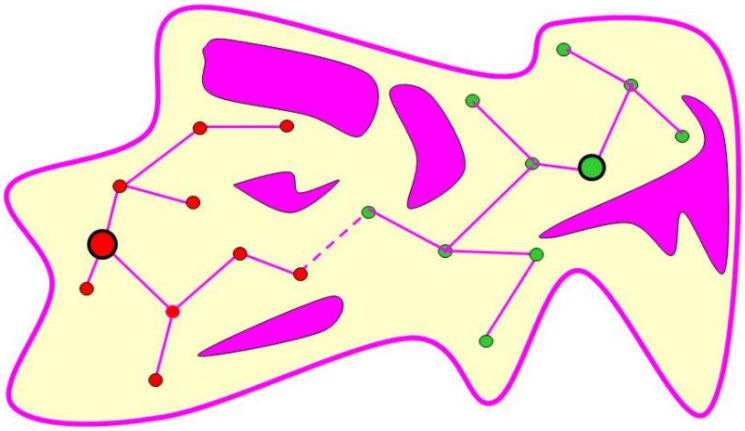

Figura 2-4. PRM de consulta única y bidireccional.

En los PRM de múltiple consulta pueden coexistir varios grafos y su planificador consta de dos fases: una fase de construcción y una fase de consulta. En la fase de construcción, se construye una hoja de ruta basada en nodos (puntos en el espacio de configuraciones) que aproxima los movimientos que se pueden realizar en el entorno. Primero, se crea una configuración aleatoria y luego se conectan algunos vecinos, normalmente los *k* vecinos más cercanos o todos los vecinos dentro de un perímetro predeterminado. Las configuraciones y conexiones se agregan al gráfico hasta que la hoja de ruta sea lo suficientemente densa. En la fase de consulta, las configuraciones de inicio y de objetivo están conectadas al gráfico, y la ruta se obtiene mediante la consulta de ruta más corta, utilizando el algoritmo de Dijkstra.

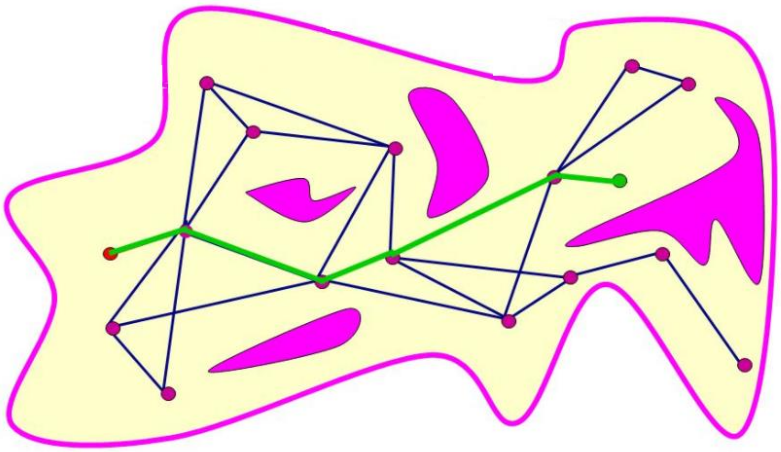

Figura 2-5. PRM de múltiple consulta.

El PRM es probabilísticamente completo, lo que significa que a medida que el número de puntos muestreados aumenta, la probabilidad de que el algoritmo no encuentre una ruta, si existe, se acerca a cero. La tasa de convergencia depende de ciertas propiedades de visibilidad del espacio libre, donde la visibilidad está determinada por el planificador local.

La principal ventaja de los PRM reside en que no es necesario calcular los límites de los C-obstáculos, y por ende de C_{free}, tarea nada fácil cuando el número de dimensiones del problema es elevado; sino que basta con utilizar un algoritmo que compruebe si existe o no colisión. En la mayoría de los casos esto resulta bastante más práctico.

2.3.2 - Algoritmos RRT

Steven M. LaValle en 1998 publicó el primer artículo en el que se describe la forma básica de un algoritmo de planificación bautizado con el nombre de **"Rapidly Exploring Random Tree",** abreviadamente RRT (LaValle, 1998). Inicialmente el RRT se ideó tanto como planificador independiente y como estructura algorítmica a utilizar en otros métodos que pudieran precisar algún tipo de proceso de exploración aleatoria.

2.3.2.1 - RRT básico

El algoritmo RRT originario se basa en la construcción de un árbol de configuraciones que crece explorando a partir de un punto origen. El objetivo original del método *RRT* consiste en construir un árbol de exploración que cubra uniformemente todo el espacio de configuraciones libres de colisión (C_{free}). Para entender mejor el funcionamiento del algoritmo, se usarán los siguientes conceptos:

$\boldsymbol{q}_{ini}$→ Configuración inicial (en el caso de un robot que se mueve en un plano, se proporcionan las coordenadas *x* e *y,* y la orientación del vehículo respecto a uno de los ejes del sistema de referencia).

$\boldsymbol{q}_{fin}$ → Configuración final que se desea alcanzar.

$\boldsymbol{q}_{rand}$ → Nodo aleatorio que genera el algoritmo dentro del espacio de configuraciones ($q_{rand} \in \mathcal{C}$).

$\boldsymbol{q}_{near}$ → Es el nodo más próximo a q_{rand}.

q_{new}→Es la configuración (nodo) que se va a añadir al árbol.

ε→ Longitud del segmento de crecimiento. Es la distancia entre un nodo del árbol y el siguiente con el que está conectado.

Nodo padre→ Nodo a partir del cual aparece otro nodo.

Como se observa en la siguiente figura, primero se define la posición inicial y final. Seguidamente se define la longitud del segmento de crecimiento.

Figura 2-6. Inicio del algoritmo RRT. Definición de nodo inicial y final.

También se pueden definir los obstáculos y el número de iteraciones en este punto. A continuación se calcula un punto aleatorio en el espacio de configuraciones y se busca el nodo más cercano a este punto.

Figura 2-7. Cálculo del nodo aleatorio.

Si la distancia entre el nodo q_{rand} y q_{near} es igual o más pequeña que ε, entonces el nuevo nodo q_{new} será igual a q_{rand}. Si por el contrario la distancia es mayor, q_{new} se calcula partiendo desde q_{near} y en dirección a q_{rand}, con una longitud total igual a la ε establecida en el paso 1.

Figura 2-8. Creación de un nuevo nodo a partir de q_rand.

Para acabar, se observa si hay colisión entre el nodo q_{near} y q_{new}. Si no hay colisión, se añade definitivamente q_{new} al árbol y vuelve a empezar la iteración. Por el contrario, si hay colisión, se vuelve a empezar la secuencia volviendo a calcular q_{rand}, pero sin añadir el nodo q_{new} previamente calculado.

Figura 2-9. Incorporación del nuevo nodo al árbol.

En el caso de la figura anterior, el padre de q_{new} es q_{ini}, siendo q_{ini} el nodo nº 0 y su hijo el nodo nº1. A medida que el árbol crece, las complejidades entre nodo padre e hijo también. Así, por ejemplo, el nuevo nodo nº 54 podría ser el hijo del nodo nº 11. De este modo, el árbol va creciendo poco a poco y acercándose a su objetivo.

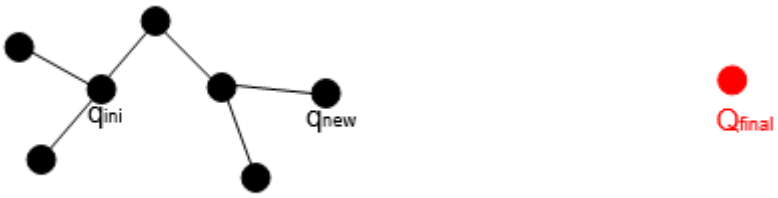

Figura 2-10. Ejemplo de posible crecimiento del árbol.

En la siguiente figura se muestra de manera simplificada cómo sería el algoritmo RRT escrito.

Algoritmo RRT

1. ***Iniciar árbol***
   Definir q_{ini}, q_{fin}, ε

For i=1 to N

2. ***Se define q_{rand}***
3. ***Se busca el nodo más cercano q_{near}***

IF [distancia (q_{near}-q_{rand}) < ε)] THEN

4. ***q_{new}=q_{rand}***

ELSE

5. ***q_{new}=New_config(q_{near},q_{rand})***

END

IF [NoObstacle (q_{new}-q_{near})] THEN

6. ***Insertar q_{new}***

END
END

Figura 2-11. Algoritmo del RRT básico.

Una vez visto el funcionamiento del algoritmo RRT, se expone una lista de las principales características de este:

- La expansión de los algoritmos basados en RRT se inclina decididamente hacia los espacios inexplorados.
- La distribución de sus vértices aproxima la de la función de probabilidad utilizada.
- Son simples, facilitando el análisis de comportamiento y la implementación en cualquier escenario.
- Siempre permanecen conexos, aún con pocos vértices.
- Un RRT es un módulo que puede ser implementado en otros planificadores.
- No requieren la existencia del par origen-destino, con lo que su ámbito de aplicación se diversifica.
- No requieren una definición explícita de C_{free}, sólo utiliza una función que comprueba la existencia de colisión.

- Es probabilísticamente completo (la probabilidad de encontrar un camino si existe tiende a 1 exponencialmente con el número de nodos).

En las siguientes figuras se muestra el funcionamiento del RRT. En la primera figura el algoritmo funciona en un entorno circular y sin obstáculos (si no se tiene en cuenta las limitaciones del gráfico). En la segunda figura el entorno es asimétrico y con obstáculos. En ambos casos, a medida que aumenta el número de iteraciones, el algoritmo ocupa el mayor espacio libre posible.

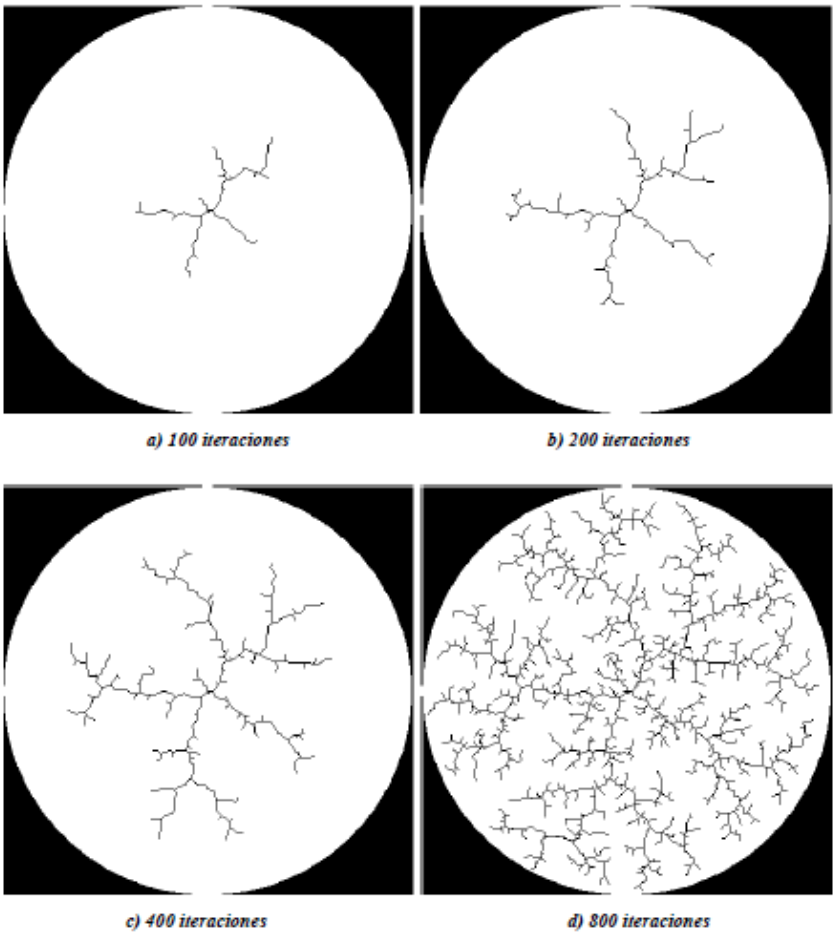

Figura 2-12. Evolución del algoritmo RRT en un entorno circular.

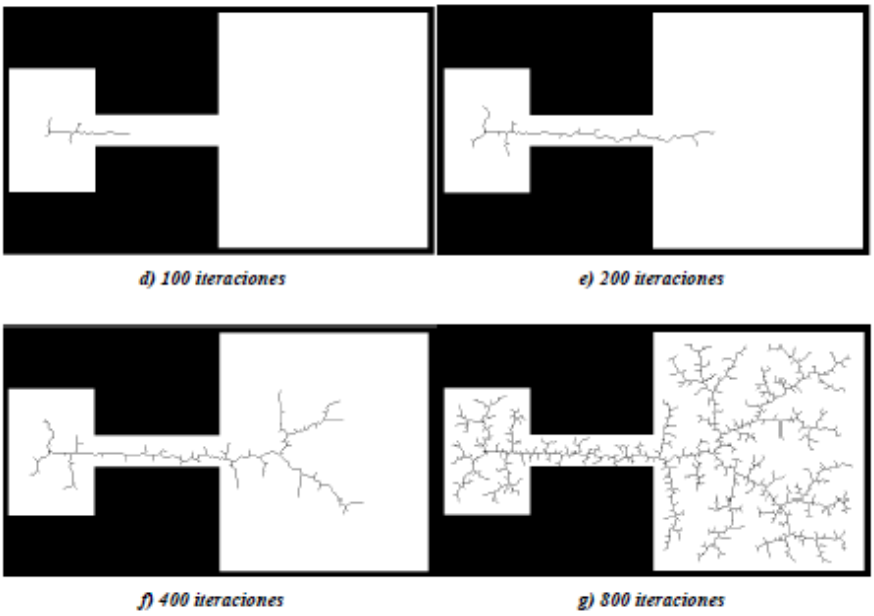

Figura 2-13. Evolución del algoritmo RRT en un entorno asimétrico.

2.3.2.2 - RRT Estrella

El algoritmo RRT* o RRT estrella sigue la misma dinámica que el RRT, pero introduce una pequeña modificación la cual optimiza el coste de la solución, aunque esto pueda hacer que el tiempo de ejecución sea más elevado.

La principal diferencia se encuentra en el momento en que el algoritmo encuentra el nodo q_{new}. Para la explicación se utilizará la misma configuración que en el ejemplo del algoritmo RRT básico. Como se observa en la siguiente figura, el árbol ya consta de 2 nodos.

Figura 2-14. Algortimo RRT* con 2 nodos.

A continuación se calcula el nodo aleatorio q_{rand}, se busca el nodo más cercano a éste y se le llama q_{near}.

Figura 2-15. Creación de nodo q_{rand} en RRT*.

Si la distancia entre el nodo q_{rand} y q_{near} es igual o más pequeña que ε entonces el nuevo nodo q_{new} será igual a q_{rand}. Si por el contrario la distancia es mayor, q_{new} se calcula partiendo desde q_{near} y en dirección a q_{rand}, con una longitud total igual a ε la cual se ha establecido con anterioridad.

Figura 2-16. Creación de nuevo nodo a partir de q_{rand}

Se observa que no haya ningún obstáculo entre q_{near} y q_{new}. Si hay algún obstáculo, no se añade el nodo q_{new} y vuelve a empezar la iteración calculando otro nodo q_{rand}. En cambio, si no se encuentra obstáculo, se añade el nodo q_{new} (nodo nº 2) al árbol y se le asigna su padre (nodo nº 1 en este caso).

Figura 2-17. Inserción de nuevo nodo al árbol.

Hasta aquí, el algoritmo RRT* es exactamente igual al RRT básico.

Primero se calcula el coste para llegar al último nodo desde el inicio. Suponiendo ε=5, el coste de ir del nodo 0 al 2 sería de 10. Seguidamente, desde q_{new} se buscan todos los nodos dentro de un perímetro con radio r (estos nodos pasarán a llamarse $q_{nearest}$).

Figura 2-18. Radio de búsqueda de los nodos $q_{nearest}$.

Una vez se tienen todos los nodos $q_{nearest}$, se calcula el coste de ir de $q_{nearest}$ a q_{new} (siempre que no haya colisión), y si este es inferior al coste inicial se actualiza el padre del nodo q_{new}.

Como se observa en la siguiente figura, el coste de ir de q_{ini} (nodo 0) a q_{new} (nodo 2) es inferior al coste inicial. Ahora el padre del nodo 2 será el nodo 0.

Figura 2-19. Re-cálculo del padre de nodo q_{new}.

Visualmente el árbol será el mismo que en el algoritmo RRT, pero a la hora de encontrar la solución, el algoritmo tendrá en cuenta los padres e hijos de cada nodo. Se puede añadir una modificación al árbol que muestre la relación padre-hijo de los nodos actualizados, como se muestra a continuación.

Figura 2-20. Cambio del padre de nodo q_{new}.

En la siguiente figura se muestra de manera simplificada cómo sería el algoritmo RRT estrella escrito.

Algoritmo RRT*

1. ***Iniciar árbol***
   Definir q_{ini}, q_{fin}, ε.

For i=1 to N

2. ***Se define q_{rand}***
3. ***Se busca el nodo más cercano q_{near}***

IF [distancia (q_{near}-q_{rand}) < ε)] THEN

4. ***q_{new}=q_{rand}***

ELSE

5. ***q_{new}=New_config(q_{near},q_{rand})***

END

6. ***Definir nodos $q_{nearest}$***

For i=1 to length (nodos $q_{nearest}$)

IF [distancia (q_{near}-q_{new})> distancia ($q_{nearest}$-q_{new})] AND [NoObstacle (q_{new}-$q_{nearest}$)] THEN

7. ***Actualizar el nodo padre de q_{new}***

END
END

8. ***Insertar nodo q_{new}***

END

Figura 2-21. Algoritmo del RRT estrella.

2.3.2.3 - RRT Bidireccional

Este algoritmo incluye una simple modificación, la cual consiste en la creación de dos árboles en vez de uno. En este caso los puntos q_{ini} y q_{fin} serán los nodos 0 de cada árbol. Sin embargo, el crecimiento del árbol A es dependiente del árbol B y viceversa.

Una vez se genera un punto aleatorio, el primer árbol (A) crece hacia ese punto, si es posible. Si ha habido una nueva rama, entonces existe un nodo nuevo q_{new}. Ahora el segundo árbol no tomará como meta el punto aleatorio q_{rand}, sino el nuevo nodo generado q_{new}. De esta forma dicho árbol no explora el espacio vacío, sino que intenta encontrar a su homólogo.

En la siguiente figura se observa una simplificación del algoritmo.

Algoritmo RRT Bidireccional

1. ***Iniciar árbol***
   Arbol_a [0] = qini
   Arbol_b [0] = qfin

For i=1 to N

2. ***q_{rand}= Configuración_Aleatoria ( );***

IF ***[Extiende (Arbol_a, q_{rand})=true]*** *THEN*

IF ***[Extiende (Arbol_b, q_{new})=true]*** *THEN*

3. ***Devuelve (Camino (Arbol_a,Arbol_b))***

4. ***Intercambiar(Arbol_a, Arbol_b)***

Next N

END

Figura 2-22. Algoritmo simplificado del RRT bidireccional.

La función *Intercambiar* alterna el orden de los árboles de manera que el reparto de sendas funciones sea equilibrado. Así pues, las acciones resultantes de dos iteraciones consecutivas serían:

1. Árbol A crece hacia q_{rand} (1a iteración).
2. Árbol B crece hacia q_{new} de A.
3. Árbol B crece hacia q_{rand} (2a iteración).
4. Árbol A crece hacia q_{new} de B.

De esta forma cada árbol invierte la mitad de su tiempo en explorar el espacio libre, y la otra mitad, en buscar a su compañero.

Una de las grandes ventajas de este algoritmo, además de su rapidez, consiste en la posibilidad que ofrece de implementarse en otros algoritmos RRT ya diseñados. Incluso si los algoritmos de planificación no son del tipo RRT, la idea también se puede extrapolar a otras soluciones.

2.4 - Conclusiones

Existe un gran abanico de diferentes métodos de planificación de movimiento. De los que se han mencionado en los apartados anteriores, existen diferencias que los hacen más o menos adecuados según el escenario dónde tengan que actuar.

La mayoría requieren de algún tipo de pre-procesamiento, el cual normalmente se basa en la definición poligonal de los obstáculos-C o la subdivisión de todo el escenario de actuación. Otros en cambio evitan esta labor previa y permiten atacar de forma directa el problema, como en los PRMs o RRTs. Sin embargo, al no necesitar un pre-procesamiento, estos últimos algoritmos tienen un coste computacional más pesado en su proceso. Según los requerimientos del problema, puede ser más interesante un método u otro en función de su velocidad, y la calidad de la solución obtenida.

Para solventarlo los problemas de pre-procesado se idearon los métodos de exploración aleatoria. Éstos tuvieron su mayor difusión dentro de los PRMs, que basaban su expansión en un muestreo probabilístico de C_{free}. El éxito de los métodos aleatorios hizo que se incrementara el estudio sobre éstos, desplazando así a los métodos *roadmap*. Sin embargo, el coste computacional y la velocidad de cálculo de los métodos aleatorios seguían siendo un problema. Para competir en velocidad y suplir la carencia de una inteligencia en la búsqueda de caminos (pre-procesado) se diseñó un algoritmo aún más simple, el RRT.

El *Rapidly Random Exploring Trees* (RRT) es inmune al número de vértices que tengan los obstáculos o a la forma de los mismos pues tan sólo necesita una función que compruebe la existencia o no de colisión. Es decir, no necesita de pre-procesado. Al mismo tiempo el RRT asegura una exploración equiprobable de todo el espacio de configuraciones, compitiendo en esta ventaja con los PRMs al tener un coste computacional menor. Mientras los PRM pueden generar grafos inconexos, el RRT siempre mantiene sus vértices conectados. Además, es sencillo de implementar, rápido y de fácil extensión a sistemas con elevado número de grados de libertad.

Debido a estas principales ventajas se ha elegido realizar este proyecto con un algoritmo RRT. Concretamente, se ha escogido el algoritmo RRT estrella o RRT*. La relativa sencillez del algoritmo RRT permite que éste pueda modificarse fácilmente. El RRT* ofrece soluciones más óptimas y sencillas que el RRT básico, ya que siempre recalcula

si hay un camino más corto. Sin embargo, esto implica un mayor coste computacional. Aun así, en nuestro caso la ventaja que ofrece el RRT* sobre el RRT es mucho mayor que su desventaja.

3 - DISEÑO DEL ALGORITMO

En este apartado se diseñará el algoritmo de planificación escogido y se comparará con otros métodos similares. Así, se observarán las principales diferencias entre los algoritmos RRT y PRM.

3.1 - RRT vs RRT*

A continuación se diseñan los algoritmos RRT vistos previamente y se comparan los resultados.

Primeramente, se realiza el RRT básico y luego el RRT*, ya que éste último es una ampliación del primero.

3.1.1 - RRT vs RRT* sin obstáculos

Para la primera comparación no se utilizará ningún obstáculo. Los parámetros que se cambiarán para ajustar la calidad del algoritmo se observan en la tabla 1.

Tabla 1. *Parámetros asociados a cada algoritmo RRT.*

RRT	RRT*
ε =Longitud del segmento de crecimiento	ε =Longitud del segmento de crecimiento
Nº de iteraciones	Nº de iteraciones
	Radio de búsqueda de $q_{nearest}$

En los siguientes resultados los parámetros q_{ini} y q_{goal} no se han modificado, siendo estos (0,0) y (800,800) respectivamente. Por lo que respecta a los nodos q_{rand}, estos estarán comprendidos entre (0,0) y (1000,1000).

Hay que tener en cuenta que con los mismos parámetros se pueden obtener diferentes soluciones, por lo que los siguientes gráficos podrían ser diferentes cada vez. Para evitar datos erróneos, se han realizado varias pruebas con los mismos parámetros y se ha hecho una media con los resultados obtenidos.

- **Estudio con diferentes ε**

En los siguientes gráficos el nº de iteraciones y el radio de búsqueda de $q_{nearest}$ (r_q) serán siempre 1000 y 50 respectivamente.

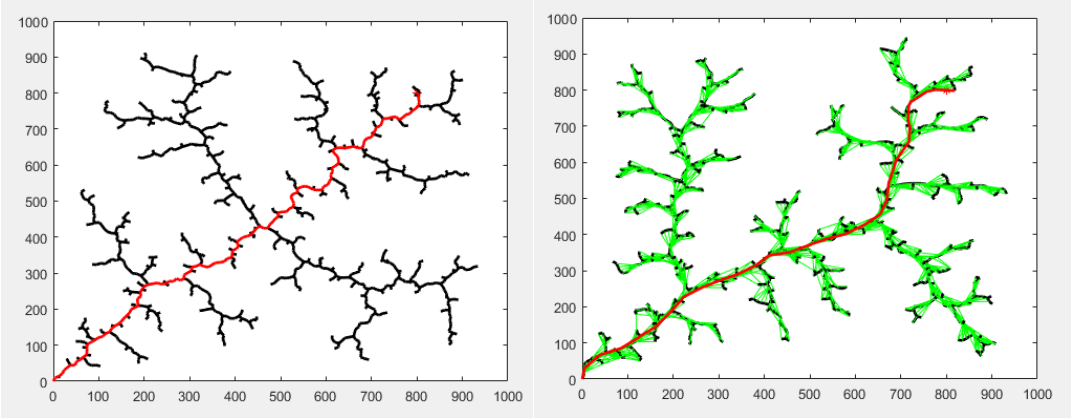

Figura 3-1. RRT (izquierda) y RRT* (derecha) con ε=10.

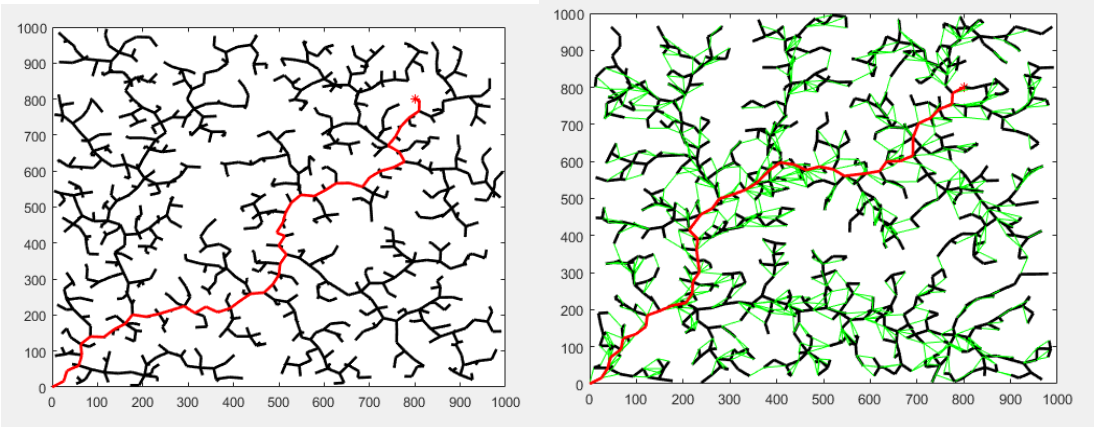

Figura 3-2. RRT (izquierda) y RRT* (derecha) con ε=30.

Las líneas verdes del algoritmo RRT* son los “atajos” que encuentra el algoritmo dentro del radio especificado.

Observando los gráficos se puede ver como el coste para llegar a la posición final es casi siempre menor en el algoritmo RRT*. Hay que tener en cuenta la longitud del factor de crecimiento, ya que si es demasiado pequeño para el nº de iteraciones especificado, no abarcará el espacio suficiente. Por el contrario, si es demasiado grande, las ramas del árbol se cruzarán entre sí y esto podría causar problemas dependiendo del sistema a estudiar. A continuación se muestran los resultados obtenidos:

Tabla 2. *Promedio del coste total del RRT con 1000 iteraciones y diferentes valores de* ε.

ε	Coste total
1	No encuentra solución
10	1386
30	1440
60	1430
120	1640
500	1335

Tabla 3. *Promedio del coste total del RRT* con 1000 iteraciones,*r_q*=50 y diferentes valores de* ε

ε	Coste total
1	No encuentra solución
10	1276
30	1373
60	1327
120	1412
500	1250

En este caso, para 1000 iteraciones, aunque los mejores resultados se den con una ε=500 y ε=10, se consideran mejor los valores de 30 ó 60, ya que con 500 las ramas se cruzan entre sí y con 10 hay mucho espacio libre sin abarcar.

- **Estudio con diferente nº de iteraciones**

En los siguientes gráficos la longitud del segmento de crecimiento y el radio de búsqueda de $q_{nearest}$ (r_q) serán siempre 30 y 50 respectivamente.

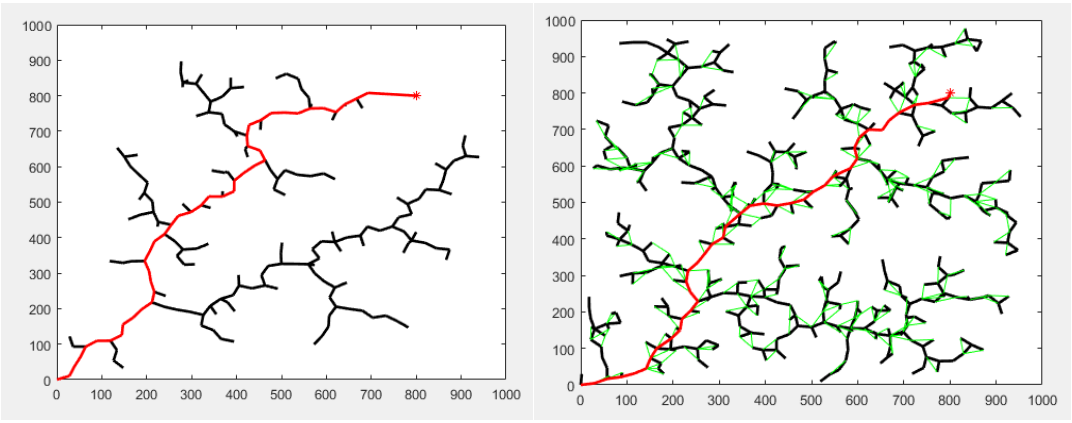

Figura 3-3. RRT (izquierda) y RRT* (derecha) con 500 iteraciones.

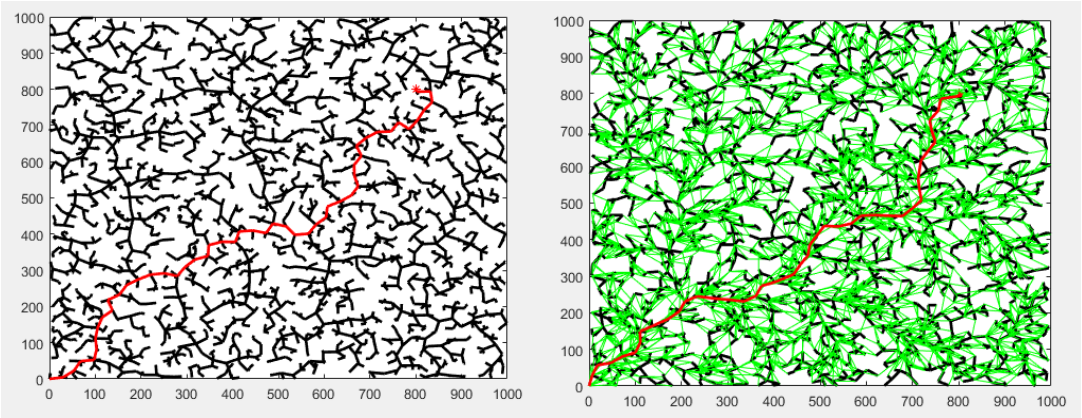

Figura 3-4. RRT (izquierda) y RRT* (derecha) con 3000 iteraciones.

En este caso el algoritmo RRT* también tiene un menor coste que el RRT. Sin embargo, al aumentar el nº de iteraciones no se aprecia un cambio tan significativo en lo que respecta al coste total. El cambio que más se puede apreciar es el espacio libre que se abarca. Los resultados obtenidos son los siguientes:

Tabla 4. *Promedio del coste total del RRT con ε=50 y diferentes nº de iteraciones.*

Nº iteraciones	Coste total
100	No encuentra solución
500	1397
1000	1440
2000	1351
3000	1444

Tabla 5. *Promedio del coste total del RRT* con ε=50, r_q=50 y diferentes nº de iteraciones*

Nº iteraciones	Coste total
100	No encuentra solución
500	1286
1000	1373
2000	1305
3000	1351

- **Estudio con diferente radio de búsqueda de $q_{nearest}$**

En los siguientes gráficos la longitud del segmento de crecimiento y el nº de iteraciones serán siempre 30 y 1000 respectivamente.

Ya que este parámetro sólo afecta al RRT estrella, solo se estudiará este algoritmo.

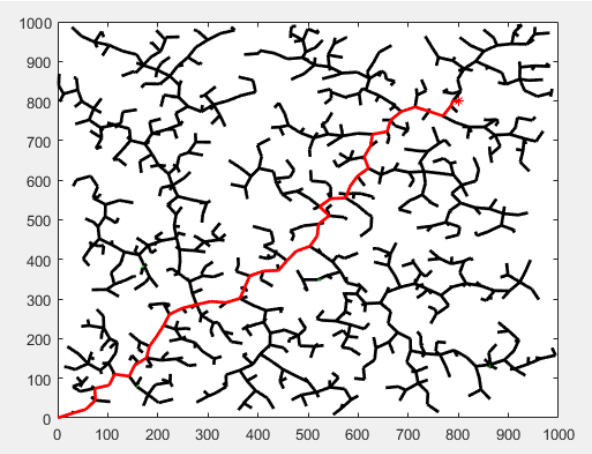

Figura 3-5. RRT* con r_q=10.

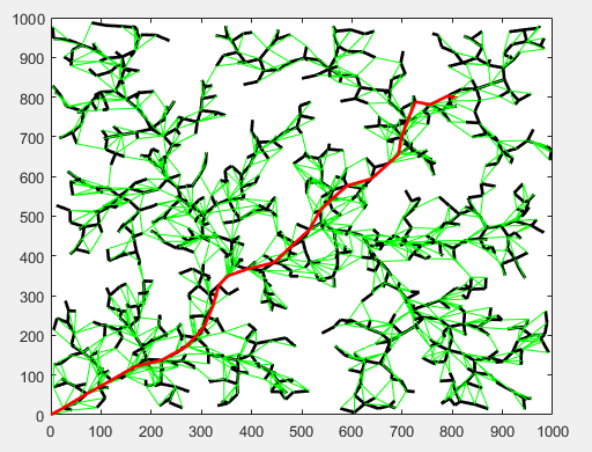

Figura 3-6. RRT* con r_q=60.

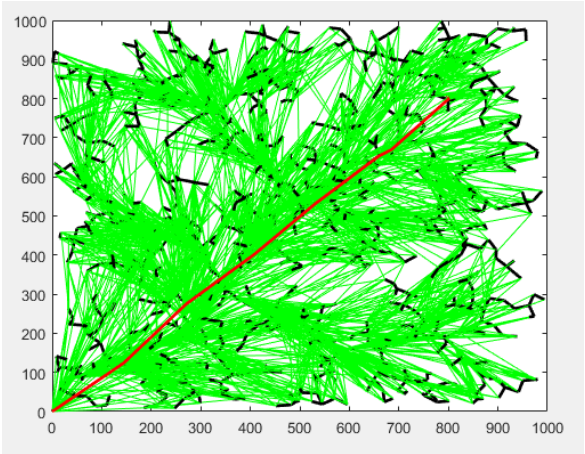

Figura 3-7. RRT* con r_q=200.

En la siguiente tabla se muestran los resultados obtenidos al cambiar el radio de búsqueda:

Tabla 6. *Promedio del coste total del RRT* con 1000 iteraciones, ε=30 y diferentes r_q.*

Rq	Coste total
10	1344
30	1330
60	1216
200	1138

Al cambiar el radio de búsqueda, sí que se puede observar una mejora significativa en lo que respecta a la búsqueda de la solución óptima.

Si r<ε el algoritmo RRT* prácticamente se convierte en un RRT básico. Sin embargo, a medida que ésta r se aumenta, la probabilidad de encontrar una mejor solución también aumenta. La única desventaja que se puede apreciar al aumentar éste valor es el aumento del cálculo de la computadora.

La elección de estos parámetros va ligada a nuestras necesidades y nuestro sistema de estudio. Teniendo en cuenta esto, si se necesita mucha precisión se disminuirá ε y se aumentará r_q y el nº de iteraciones. Si por el contrario se necesita rapidez, se aumentará ε, se disminuirá $\mathbf{r_q}$ y el nº de iteraciones.

Para la sintonización de estos parámetros siempre se deberá tener en cuenta las dimensiones donde se aplique el algoritmo y la respuesta que se desee.

3.1.2 - RRT vs RRT* con obstáculos

Con la implementación de obstáculos también se deben hacer pequeños cambios en el algoritmo, ya que cabe la posibilidad de que una vez se acabe el nº de iteraciones, se llegue al punto final desde una posición en la que colisione con un obstáculo. Con la siguiente figura se observa con más claridad este problema.

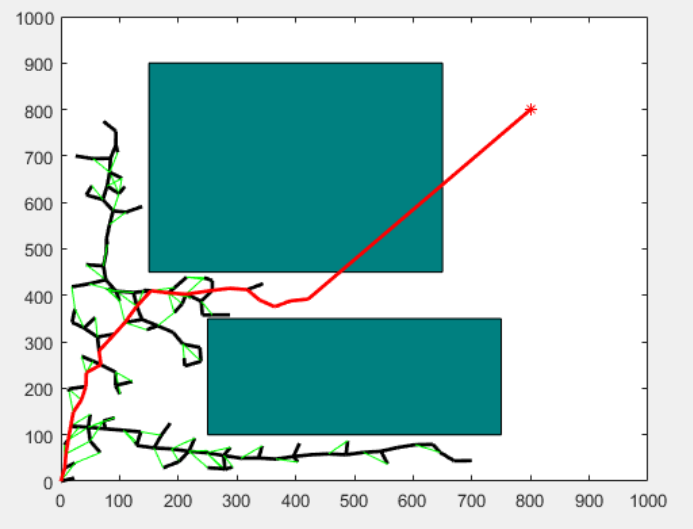

Figura 3-8. RRT*, ε= 30, iteraciones=500, r=50.

Se puede observar como en este caso la respuesta final no sería aceptable. Para solucionar este problema se ha cambiado la dinámica de búsqueda. En vez de finalizar cuando se acaben las iteraciones, el algoritmo finalizará cuando encuentre un nodo lo suficientemente cerca de q_{fin} en el cual no haya colisión. De esta manera, el nº de iteraciones siempre será diferente y si existe una solución al problema, siempre se encontrará. Por el contrario, no siempre se obtendrá el camino óptimo, sino el primer camino encontrado.

A continuación, se procede a comparar los dos algoritmos, pero esta vez con los dos obstáculos vistos en la figura anterior y con el cambio mencionado.
Se añade un nuevo parámetro “umbral”, el cual define la distancia máxima a la que se puede encontrar el nodo padre de q_{fin}, ya que si no se define ningún umbral, el algoritmo tendría que iterar muchas veces para acabar.

- **Estudio con diferentes ε**

En los siguientes gráficos el radio de búsqueda de $q_{nearest}$ (r_q) y el umbral serán siempre 60 y 10 respectivamente.

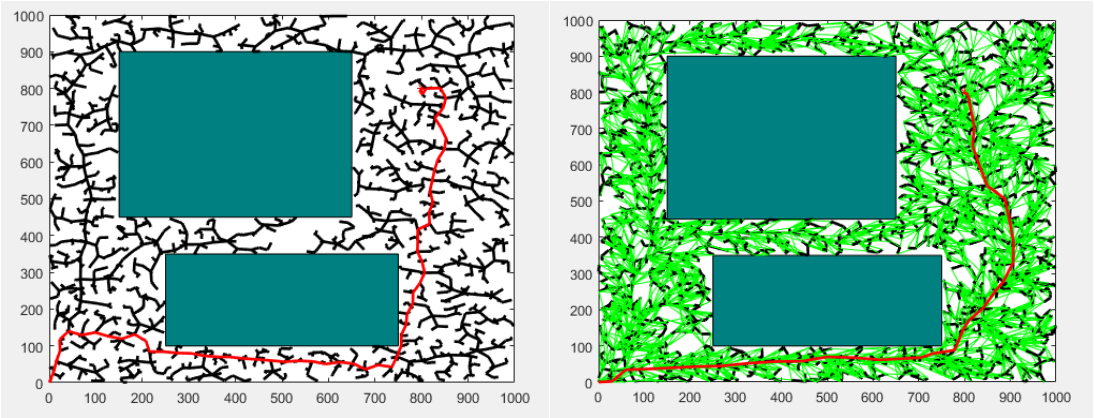

Figura 3-9. RRT (izquierda) y RRT* (derecha) con ε=10.

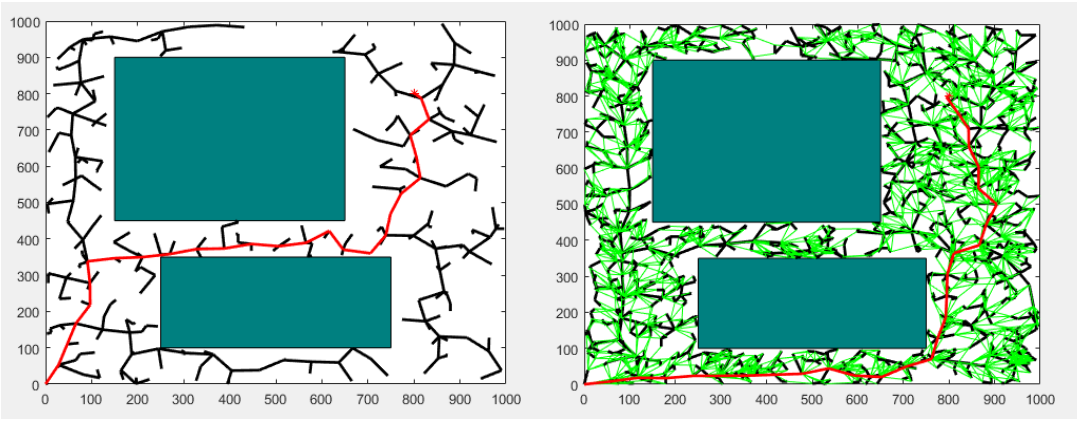

Figura 3-10. RRT (izquierda) y RRT* (derecha) con ε=60.

Se ha utilizado un umbral de 10, ya que de esta manera se obtenía una solución con relativa rapidez. Los resultados obtenidos se muestran en las siguientes tablas:

Tabla 7. *Promedio del coste total del RRT y nº de iteraciones con diferentes ε.*

ε	Coste total	Nº iteraciones
10	1768	1813
30	1571	2556
60	1520	985

Tabla 8. *Promedio del coste total del RRT* y nº de iteraciones con r_q=60 y diferentes ε.*

ε	Coste total	Nº iteraciones
10	1571	2556
30	1447	1513
60	1597	1451

Con estos resultados se observa que mayormente el algoritmo RRT necesita menos iteraciones para llegar a una solución que el algoritmo RRT estrella. También se puede observar que el nº de iteraciones necesario para encontrar una solución es inversamente proporcional al valor de **ε.**

- **Estudio con diferente radio de búsqueda de $q_{nearest}$**

En los siguientes gráficos ε y el umbral serán siempre 60 y 10 respectivamente.

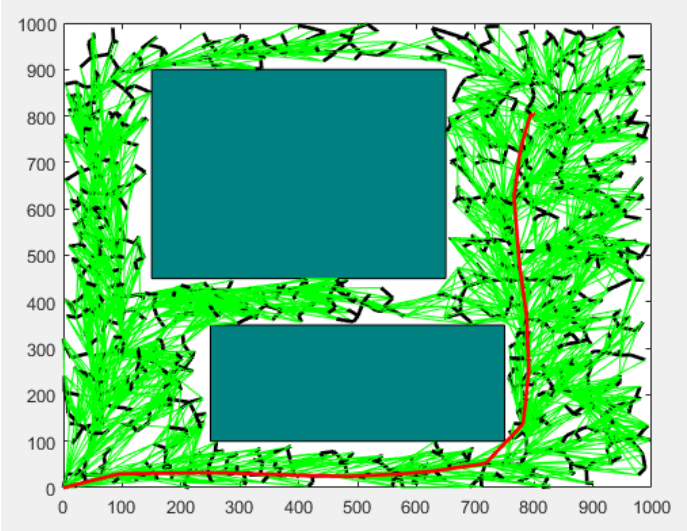

Figura 3-11. RRT* con r_q=120.

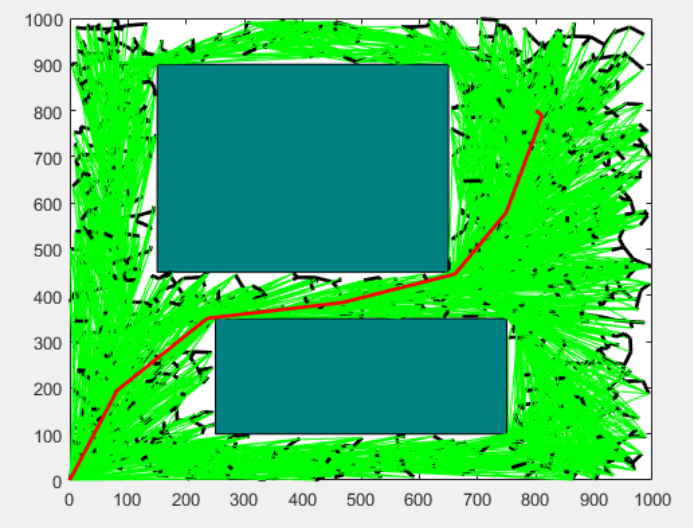

Figura 3-12. RRT* con r_q=240.

Los resultados obtenidos son los siguientes:

Tabla 9. *Promedio del coste total del RRT* y nº de iteraciones con r_q=60 y diferentes ε.*

Rq	Coste total	Nº iteraciones
60	1597	1451
120	1508	1258
240	1260	1418

Al ampliar el radio de búsqueda, permite al algoritmo evitar nodos innecesarios y atajar. Aquí se observa la gran diferencia entre estos dos algoritmos, mostrando el RRT* una respuesta más suave. Teniendo en cuenta que se deberá aplicar al control de un coche autónomo, la elección de este algoritmo frente al RRT básico parece obvia, ya que esto facilitará la respuesta que pueda dar el controlador.

3.2 - PRM vs RRT*

Para esta comparación, se ha utilizado la librería de Matlab *robotics.PRM*, la cual permite evaluar el comportamiento de un mapa probabilístico. Para el funcionamiento de esta librería solo se le debe añadir el punto inicial y final (q_{ini},q_{goal}), la localización de los obstáculos y el número de nodos existentes. Se han modificado estas entradas de tal manera que se puedan comparar con los resultados anteriores. Debido a que los PRM son más restrictivos en lo que respecta a su configuración, las comparaciones que se realizan serán respecto a la cantidad de nodos totales.

Los parámetros del RRT* que permanecerán invariables son los siguientes:

- ε=30
- Rq=240

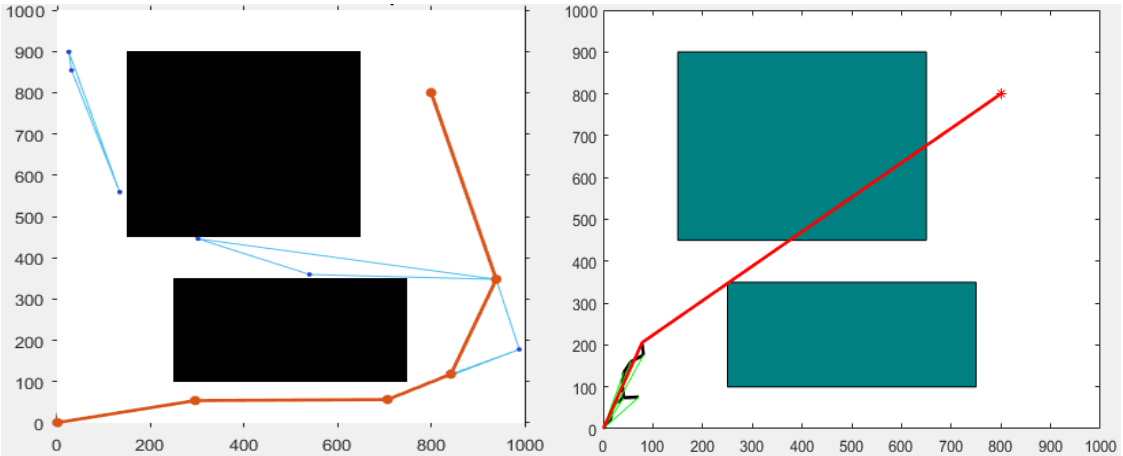

Figura 3-13. Comportamiento de PRM (izquierda) y RRT*(derecha) con 10 nodos.

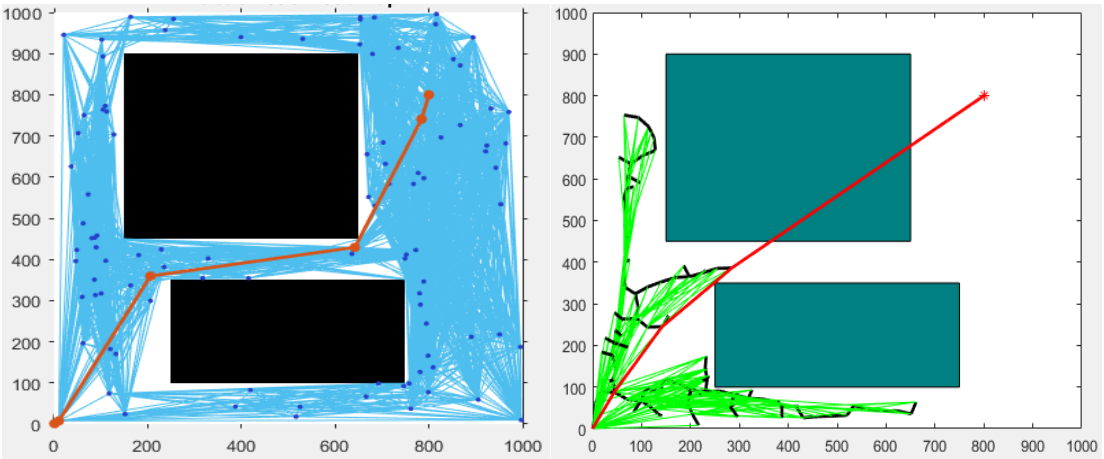

Figura 3-14. Comportamiento de PRM (izquierda) y RRT*(derecha) con 100 nodos.

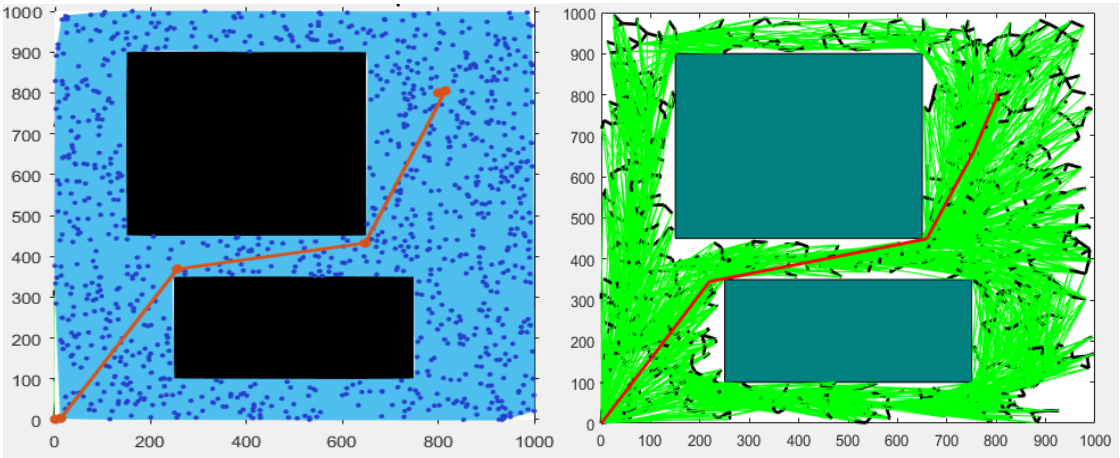

Figura 3-15. Comportamiento de PRM (izquierda) y RRT*(derecha) con 1000 nodos.

Como se observan en las figuras 3-13 y 3-14, el algoritmo PRM puede encontrar un camino válido con un nº de nodos bastante bajo, mientras que el RRT* no lo consigue encontrar. Esto se debe a que la única restricción que tienen los nodos que se añaden en el PRM, es que estos deben estar en el espacio libre de colisiones. Por el contrario, en el RRT*, estos nodos además de no colisionar con los obstáculos, deben tener un nodo padre a una distancia máxima de ε.

Esta diferencia hace que el algoritmo PRM sea más rápido que el RRT* para soluciones sencillas y con pocas restricciones dinámicas y cinemáticas. Sin embargo, es un algoritmo difícil de modificar, por lo que se debe saber con anterioridad el nº de nodos que se desea. Esta restricción hace difícil la implementación de los PRM en robots no holonómicos.

En la figura 3-15 no se observa gran diferencia en los dos resultados obtenidos. No obstante, el algoritmo PRM tiene un coste computacional mucho más elevado que el RRT estrella. Este coste se debe a su comportamiento, ya que para cada nuevo nodo que se crea, el PRM debe calcular la posibilidad de llegada a todos los nodos ya existentes. El coste computacional es exponencialmente proporcional al nº de nodos.

En conclusión, las principales desventajas del PRM respecto al RRT* se observan con mayor intensidad cuando éstos algoritmos son implementados en robots no-holonómicos o cuando se requiere de una solución óptima que no implique un tiempo computacional excesivo. Como nuestro estudio se basa principalmente en estas dos situaciones, se decide continuar con la implementación de los algoritmos RRT* en el vehículo móvil.

3.3 - Mejoras del RRT*

Los caminos obtenidos por los distintos algoritmos RRT no son siempre los más cortos ni los más sencillos. Además, éstos serían difíciles de seguir por un coche real. Dado este problema, se mejora el algoritmo añadiendo parámetros y restricciones, los cuales ayudarán a su implementación en el vehículo móvil.

3.3.1 - Parámetro θ

Para poder controlar un coche se necesitan como mínimo tres variables: (x,y,θ). Este último parámetro no nos lo puede dar el algoritmo RRT* básico. Sin embargo, se puede obtener fácilmente si se tiene un punto $n_1(x_1,y_1)$ inicial y un punto $n_2(x_2,y_2)$ final.

$$\theta = \operatorname{atan}\left(\frac{y2-y1}{x2-x1}\right) \tag{3.1}$$

Siendo n_1= (5,7) y n_2= (20,33), el ángulo deseado de llegada del coche sería de 60º aproximadamente.

Este ángulo se le añade al RRT* de manera que a la posición final se llegue con el ángulo especificado ± un umbral específico. De esta manera, el algoritmo se vuelve más restrictivo, ya que para que éste dé una solución válida ahora se tienen 3 restricciones:

- No colisionar
- Llegar a la posición deseada
- Llegar con el ángulo deseado

A partir de ahora solamente se trabajará con algoritmos RRT*. Para que los resultados sean visualmente más entendibles, se eliminarán las líneas verdes que indican los 'atajos' del RRT*. Aun así, se obtendrá el mismo comportamiento ya que solo se elimina la parte gráfica.

A continuación se muestran varios resultados con los mismos parámetros pero variando el ángulo deseado:

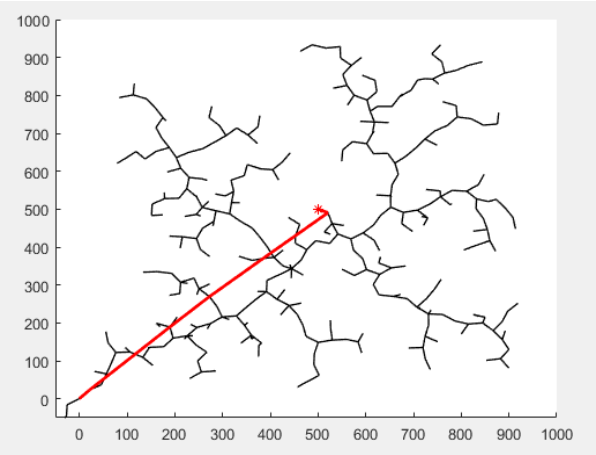

Figura 3-16. Algoritmo RRT* con un ángulo de llegada de 130º.

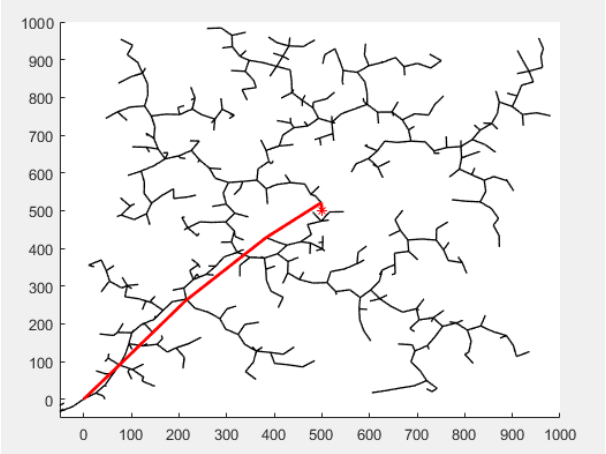

Figura 3-17. Algoritmo RRT* con un ángulo de llegada de 270º.

De esta manera se puede obtener un resultado final más exacto, ya que así se podrá definir la posición y ángulo de llegada del vehículo.

Los resultados finales muestran un giro muy brusco para poder llegar a q_{fin}, ya que solamente se ha definido el ángulo de llegada en el último nodo. Para evitar estos cambios, los cuales no serían posibles en el sistema físico, se ha implementado la mejora que se explicará a continuación.

3.3.2 - División del problema

Si al programa solamente se le facilita el punto inicial y el punto final, éste encontrará una solución. Sin embargo, puede ser que el comportamiento no sea el deseado y que nuestro vehículo realice movimientos o recorridos complicados e innecesarios.

Para solucionar esto, se divide el problema de manera que no haya un solo nodo final (q_{fin}), sino varios. De esta manera, como se observa en la figura 3-18, el nodo inicial será "A" y el nodo final "A1". Una vez se haya llegado a "A1", éste será el nodo inicial y "A2" será el nodo final. Así se conseguirá llegar al nodo final "B" disminuyendo las posibilidades de que la ruta planificada ofrezca comportamientos no deseados.

Estos puntos de referencia vienen dados por otro algoritmo el cuál calcula los puntos de paso óptimos. Sin embargo, al no disponer de éste algoritmo, en nuestro caso los puntos de referencia se impondrán y por lo tanto, puede ser que no sean los óptimos. Aun así, el sistema deberá funcionar perfectamente.

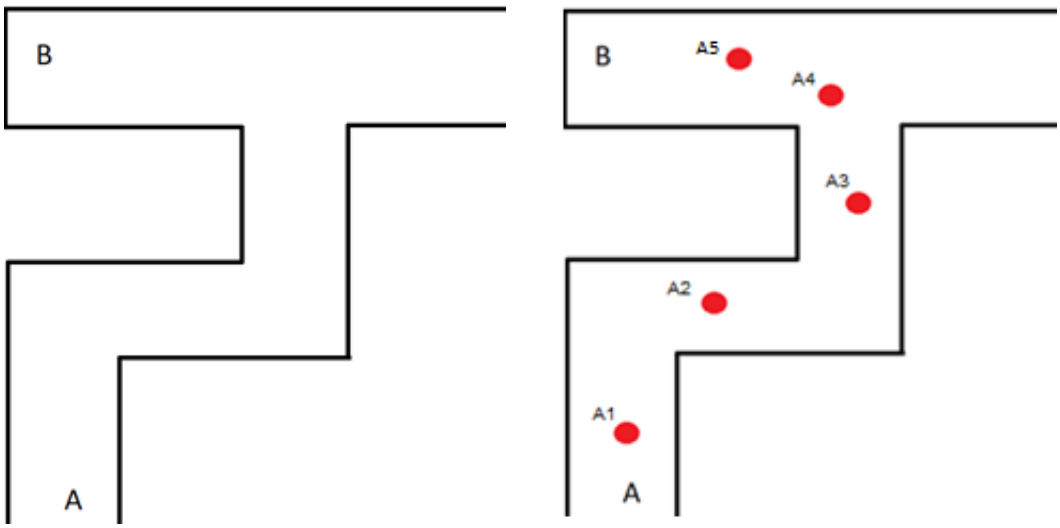

Figura 3-18. Croquis del problema a resolver (izquierda). Croquis del problema a resolver mediante la subdivisión de éste (derecha).

3.3.3 - Simplificación geométrica

Para poder implementar las anteriores mejoras se debe restringir la búsqueda de q_{goal}. Una vez se tienen todos los puntos de referencia (coordenadas por donde debe pasar el coche) y sus respectivos ángulos de llegada, se puede restringir la búsqueda de estos puntos. Para ello, en cada punto de referencia se define un umbral en el cual el RRT* podrá moverse. Fuera de este umbral el RRT* detectará colisión.

Este umbral vendrá definido por dos líneas rectas a una distancia L (la cual se podrá modificar) del punto de referencia y con un ángulo igual al ángulo de llegada a ese punto. En la figura 3-19 se muestra un ejemplo del diseño de este umbral.

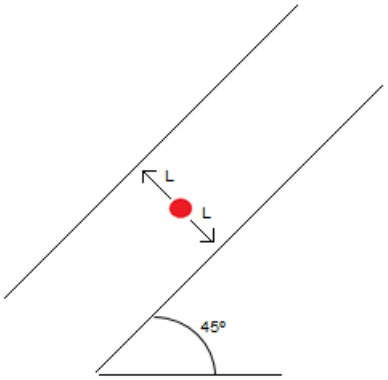

Figura 3-19. Diseño del umbral de restricción.

Para el cálculo del umbral por el cual el algoritmo podrá crecer, se deberá conocer la ecuación de la recta de este umbral. Primeramente, se debe calcular la ecuación de la recta del punto de referencia. Esta viene definida por la siguiente ecuación:

$$y - y_0 = \tan(\theta)(x - x_0) \qquad (3.2)$$

Donde x0 e y0 son las coordenadas del punto de referencia y tan θ es la pendiente. De esta manera se obtiene la ecuación de la recta que pasa por el punto de referencia:

$$y = \mathrm{mx} + \mathrm{n} \rightarrow y = \tan(\theta)\,\mathrm{x} - \tan(\theta)\,x_0 + y_0 \qquad (3.3)$$

Para obtener las ecuaciones de las rectas de los umbrales paralelos, simplemente se añadirá el término d. Esta variable se calcula mediante L y el ángulo θ y es la distancia en el eje y respecto a la recta original.

$$d = \frac{L}{cos(\theta)} \qquad (3.4)$$

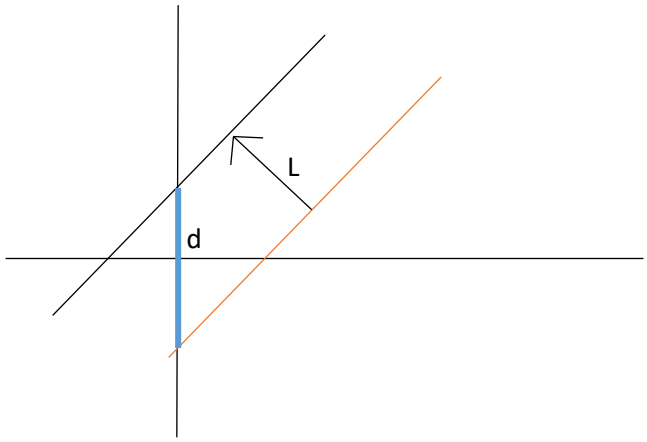

Figura 3-20. Cálculo del elemento d para la ecuación de la recta.

Las ecuaciones de las rectas los umbrales vendrán definidas por:

$$y = \tan(\theta)\,\mathrm{x} - \tan(\theta)\,x_0 + y_0 \pm d \qquad (3.5)$$

En la figura 3-21 se observa la implementación del umbral de restricción en el caso práctico visto en el apartado anterior.

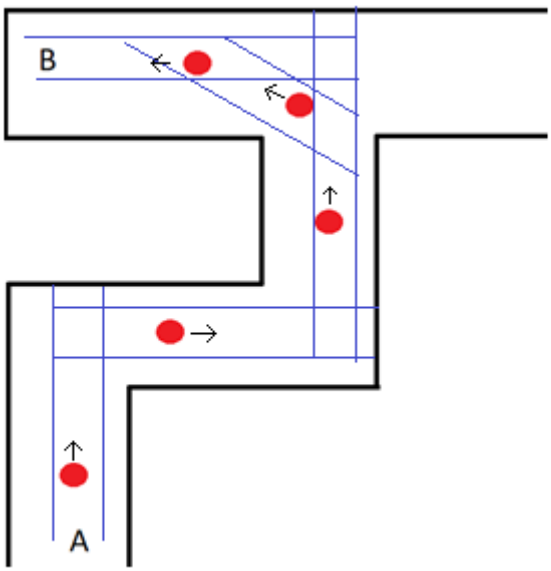

Figura 3-21. Ejemplo del problema con la implementación del umbral de restricción.

De esta manera, a la hora de detectar colisiones, el algoritmo no diferenciará si son colisiones de verdad o si son colisiones virtuales (impuestas por el umbral), y por lo tanto actuará de la misma forma.

3.3.4 - Cálculo del coste

Para llegar de un punto A a otro punto B, nuestro algoritmo calcula el coste mínimo teniendo en cuenta la distancia euclidiana que se debe recorrer entre los nodos:

$$Coste = d(P_1, P_2) + D(P_2) \quad (3.6)$$

Siendo $d(P_1,P_2)$ la distancia euclidiana entre el nodo encontrado P_1 y su padre P_2 y $D(P_2)$ el coste total con el que se llega a P_2.

El único problema que puede ofrecer este cálculo es que la distancia óptima implique cambio brusco de dirección. En el cálculo del RRT* esto no ofrece ningún problema, pero cuando se implementa el control del vehículo, esto podría dificultar el control estable del coche. Por ello, se decide utilizar otra ecuación de coste, la cual también tenga en cuenta el cambio de dirección del vehículo:

$$Coste(P, \theta) = d(P_1, P_2) \propto + C(\theta_1, \theta_2)^2 (1 - \propto) + D(P_2) \quad (3.7)$$

Donde $C(\theta_1,\theta_2)$ es la diferencia de ángulo entre el nuevo nodo encontrado y su padre y α es un parámetro ajustable. Esta diferencia se eleva al cuadrado para así diferenciar los cambios de giro que se producen de un nodo a otro. Si no se elevase al cuadrado, el algoritmo no diferenciaría cambios bruscos, ya que la suma total de ángulo girado sería la misma.

El parámetro α permite dar más importancia a la distancia o a los cambios de giro. Si α=0 solo se tendrá en cuenta las diferencias de ángulo de llegada entre nodos, y si α=1 solo se tendrá en cuenta la distancia recorrida.

De esta manera, se podrá obtener una ruta más adaptada al coche, aunque ésta no sea la más corta.

3.3.5 - Resultados

A continuación se muestran las pruebas realizadas una vez se han implementado las mejoras descritas en los apartados anteriores.

Los puntos de referencia y sus respectivos ángulos de llegada serán los siguientes:

- q_0 = (0,0) ; θ_0=0º ; Punto de inicio
- q_1 = (100,50) ; θ_1=0º
- q_2 = (250,100) ; θ_2=40º

- q_3 = (200,300) ; θ_3=100º
- q_4 = (300,350) ; θ_4=30º ; Punto de llegada

Tabla 10. *Parámetros utilizados para las pruebas de postprocesado.*

ε	R_q	Umbral	α
10	30	50	1
10	30	50	0
10	60	50	1
10	60	50	0
10	60	50	0.998

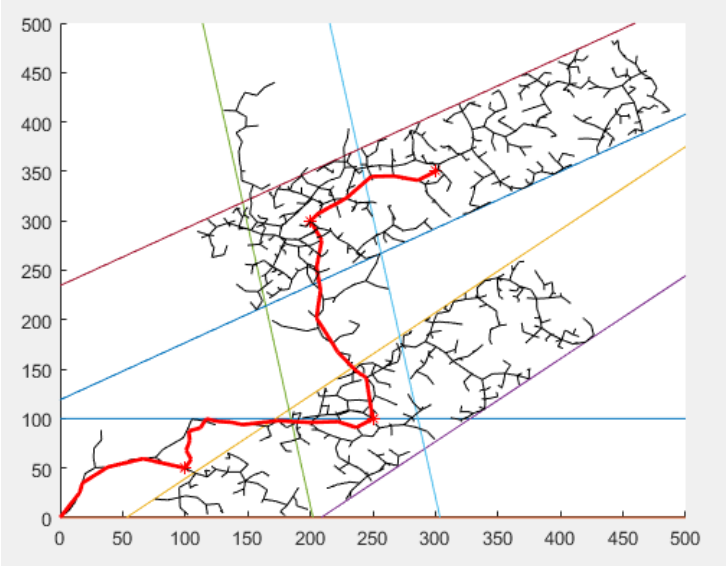

Figura 3-22. Resultado obtenido con parámetros de la fila 1, tabla 10.

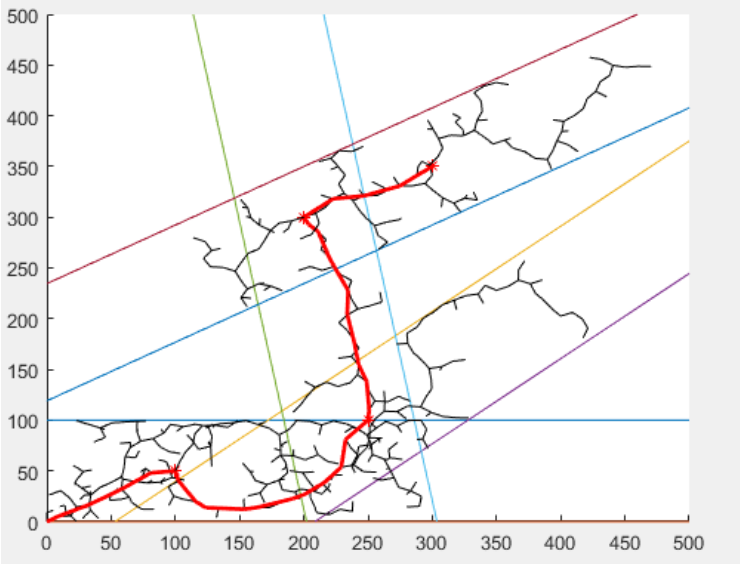

Figura 3-23. Resultado obtenido con parámetros de la fila 2, tabla 10.

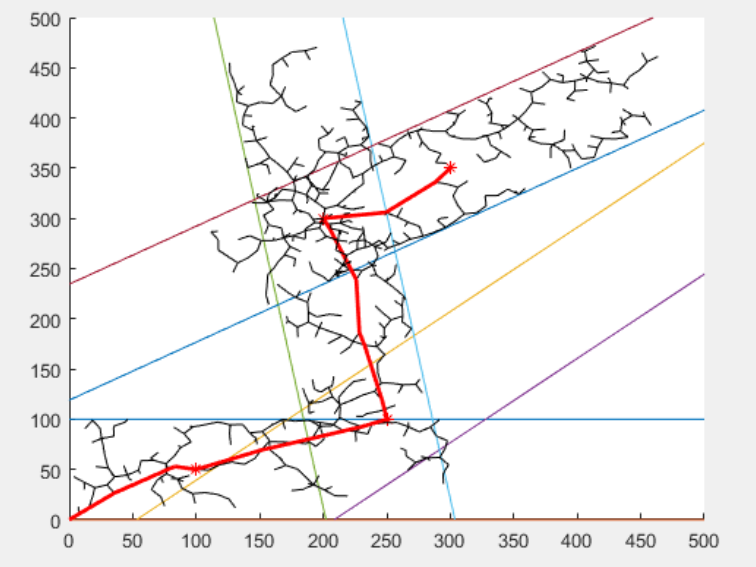

Figura 3-24. Resultado obtenido con parámetros de la fila 3, tabla 10.

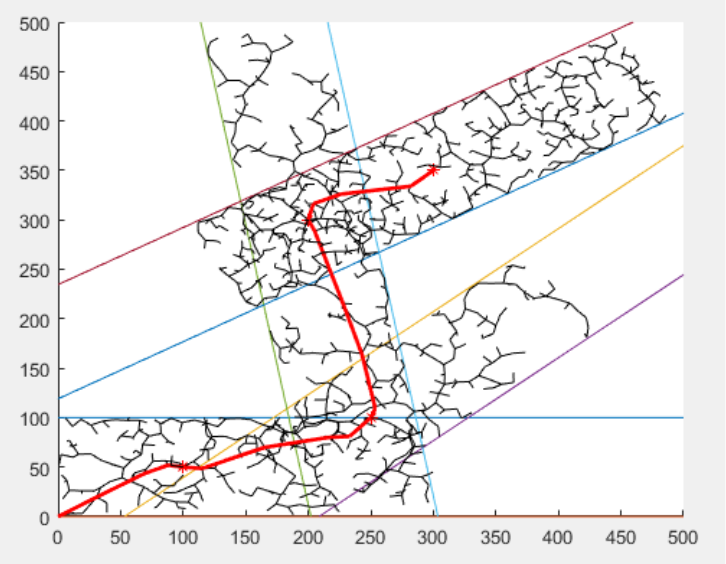

Figura 3-25. Resultado obtenido con parámetros de la fila 4, tabla 10.

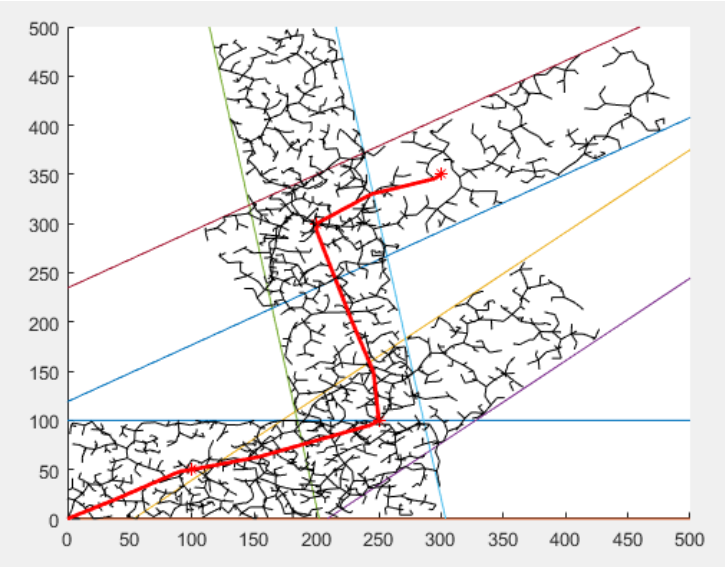

Figura 3-26. Resultado obtenido con parámetros de la fila 5, tabla 10.

Con las pruebas realizadas, se puede observar cómo afecta el parámetro α al resultado final. Si sólo se tiene en cuenta el ángulo para el coste (figura 3-23) se puede observar como el RRT* buscará el camino que menos giros bruscos contenga dentro de un radio R_q, aunque esto implique que su recorrido sea más largo. Hay que tener en cuenta que aun así, este camino puede contener giros significativos, simplemente será el mejor camino según los parámetros que se hayan especificado y según los nodos que el algoritmo haya creado.

El valor *alpha* tiene más influencia cuanto mayor sea Rq, ya que esto permite encontrar una ruta más óptima según lo especificado (camino más corto o camino más suave).

Para finalizar, se ha decidido hacer una última prueba, con un valor de α=0.998. Se escoge este valor para que el coste de distancia y de ángulo tenga un peso parecido. Si se tiene una diferencia de ángulo de 90º y una distancia de $\varepsilon=10$:

$$d(P_1, P_2) \propto= 9.98$$

$$C(\theta_1, \theta_2)^2(1-\propto) = 16.2$$

En este caso el coste del ángulo tendrá un peso mayor ya que 90º es un giro complicado para el controlador, pero si el giro es de 45º, su valor sería de 4.05, inferior al coste de la distancia. De esta manera, se diseña un sistema que valora la distancia y los cambios de giro para buscar el camino óptimo.

4 - SISTEMA DE CONTROL

Para diseñar un buen controlador el cual permita al vehículo seguir la ruta especificada por el RRT* primeramente se deben obtener las ecuaciones dinámicas y cinemáticas del vehículo y seguidamente los parámetros de este (anchura, peso, longitud, coeficiente de fricción…).

Debido a que el objetivo principal del presente trabajo es el diseño del algoritmo de planificación, no se diseñará un sistema de control demasiado complejo. Éste ignorará las ecuaciones dinámicas, simplificando así el problema.

Se diseñará un controlador *fuzzy* o controlador borroso. Se escoge este tipo de controlador porque no necesita de una matemática avanzada, es fácil de entender, flexible y puede modelar funciones no lineales.

Otro aliciente para usar este tipo de controlador es que intenta emular el comportamiento de un ser humano, por lo que para su diseño se utilizan técnicas de Inteligencia Artificial, como también se han usado para el diseño del algoritmo RRT estrella.

4.1 - Lógica difusa

El concepto de lógica difusa o lógica borrosa está asociado con la manera en que las personas perciben el medio. Por ejemplo, ideas relacionadas con el físico de una persona, la temperatura o la velocidad, cotidianamente se formulan de manera ambigua y dependiendo de quién percibe el efecto físico o químico, la respuesta a ese efecto puede variar. Cuando se dice que una persona es alta, que hace frío o que un coche va muy rápido, son afirmaciones subjetivas que dependen del observador.

Los conjuntos borrosos intentan modelar estas ambigüedades con las que se percibe una variable. Estos conjuntos son la base para la lógica difusa, del mismo modo que la teoría clásica de conjuntos es la base para la lógica Booleana. Con los conjuntos difusos se realizan afirmaciones lógicas del tipo “Si ocurre esto – Entonces pasa esto”. Este tema es propio de inteligencia artificial, donde se intenta emular el pensamiento humano.

A continuación se describen los conceptos básicos sobre la lógica difusa.

4.1.1 - Conjuntos borrosos

Los conjuntos borrosos son una extensión de los clásicos, donde varía su función de pertenencia.

Por cada conjunto o subconjunto, se define una función de pertenencia denominada ***μA(x)***, la cual indica el grado en que la variable *x* está incluida en el concepto representado por la etiqueta A *(0≤μA(x)≤1)*. Si esta función toma el valor 0 significa que el valor de *x* no está incluido en A. Si por el contrario toma el valor 1, el correspondiente valor de *x* estará absolutamente incluido en A.

Por consiguiente, un conjunto difuso A en *X*, se define como un conjunto de pares ordenados:

$$A = \{(x, \mu A(x)) / x \in X\} \qquad (4.1)$$

Donde *μA(x)* es una función de pertenencia cuya etiqueta es A y su dominio es X.

4.1.2 - Funciones de pertenencia

Las funciones de pertenencia representan el grado de pertenencia de un elemento a un conjunto, definido por una etiqueta.

Seguidamente se muestran algunas de las funciones más utilizadas:

$$A(x) = \begin{cases} 0 & si \quad x \le a \\ (x-a)/(m-a) & si \quad x \in (a,m] \\ (b-x)/(b-m) & si \quad x \in (m,b) \\ 0 & si \quad x \ge b \end{cases}$$

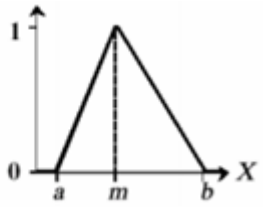

Figura 4-1. Función de pertenencia triangular.

$$A(x) = \begin{cases} 0 & si \quad x \le a \\ 2\{(x-a)/(b-a)\}^2 & si \quad x \in (a,m] \\ 1-2\{(x-a)/(b-a)\}^2 & si \quad x \in (m,b) \\ 1 & si \quad x \ge b \end{cases}$$

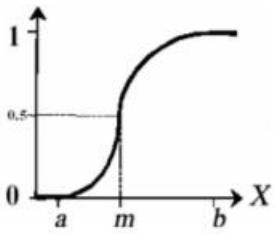

Figura 4-2. Función de pertenencia forma S.

$$A(x)=\begin{cases} 0 & si \quad (x\le a) o (x\ge d) \\ (x-a)/(b-a) & si \quad x\in(a,b] \\ 1 & si \quad x\in(b,c) \\ (d-x)/(d-c) & si \quad x\in(b,d) \end{cases}$$

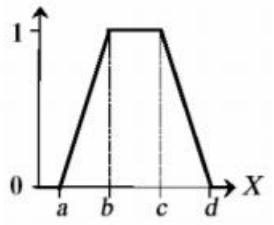

Figura 4-3. Función de pertenencia trapezoidal.

4.1.3 - Operaciones borrosas

Los subconjuntos pueden realizar operaciones entre ellos o bien se les puede aplicar determinados operadores. En ambos casos, el resultado final será un solo conjunto.

Para entender mejor estas operaciones, seguidamente se muestran algunos ejemplos básicos:

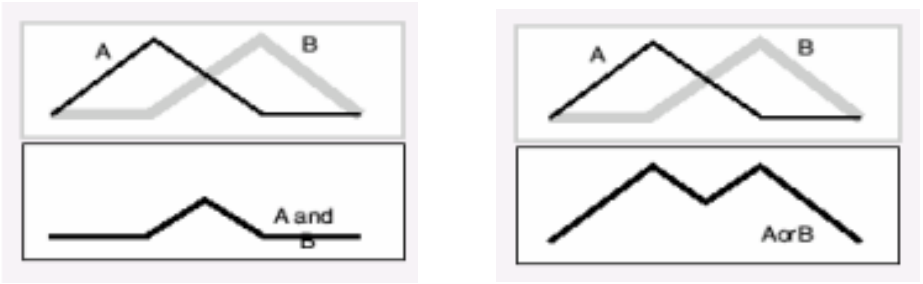

Figura 4-4. Operador Lógico AND (izquierda). Operador Lógico OR (derecha).

4.1.4 - *Fuzzificación*

En el control borroso siempre existe el proceso de *Fuzzificación*, el cual se realiza en cada ciclo que se evalúa el controlador. Es un procedimiento matemático en el que se convierte un elemento del universo de discurso (variable medida del proceso) en un valor para cada función de pertenencia a las cuales pertenece.

En la siguiente figura se muestra un ejemplo de *fuzzificación*:

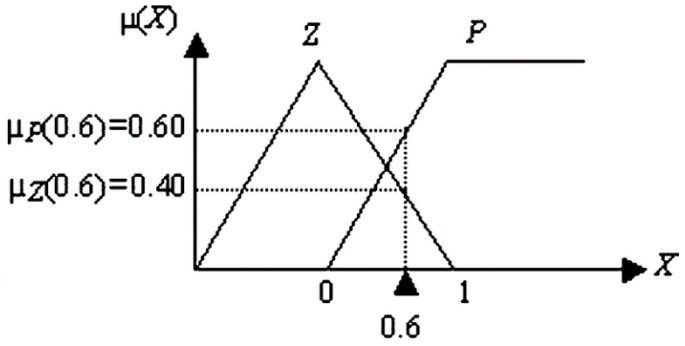

Figura 4-5. Ejemplo de fuzzificación de una variable.

Se puede observar que el valor de entrada 0.6 pertenece a los dos conjuntos, con distinto grado de pertenencia para cada uno.

4.1.5 - Reglas borrosas

Los controladores difusos usan reglas que combinan uno o más conjuntos borrosos de entrada llamados **antecedentes** o **premisas** y le asocian un conjunto borroso de salida llamado **consecuente** o **consecuencia**. A estas reglas se les llama reglas borrosas o *fuzzy rules*. Son afirmaciones del tipo SI-ENTONCES y asocian los conjuntos borrosos del antecedente mediante operaciones lógicas borrosas AND, OR, etc.

Las reglas borrosas son proposiciones que permiten expresar el conocimiento que se dispone sobre la relación entre antecedentes y consecuentes. Para expresar este conocimiento de manera completa normalmente se precisan varias reglas, que se agrupan formando lo que se conoce como **base de reglas**. Es esta base de reglas la que determina el comportamiento del controlador la que emula el comportamiento de un ser humano.

La estructura de las reglas es la misma tanto para controladores como para modelos, simplemente cambiarán las variables implementadas.

El formato general de las reglas es: **si x es A, entonces y es B.** Seguidamente se muestran ejemplos básicos de estas reglas:

- Si la presión es alta, el volumen es pequeño.
- Si la caja está vacía, el peso es pequeño.
- Si el tomate está verde, entonces no está maduro

4.1.6 - Inferencia borrosa

Las reglas difusas o borrosas representan el conocimiento y la estrategia de control, pero cuando se asigna información específica a las variables de entrada en el antecedente, la inferencia difusa es necesaria para calcular el resultado de las variables de salida del consecuente. Este resultado es en términos difusos, es decir que se obtiene un conjunto difuso de salida de cada regla, que posteriormente junto con las demás salidas de reglas se obtendrá la salida del sistema.

Existe una gran cantidad de métodos de inferencia difusa, pero los más importantes en el campo del control son los mostrados en la siguiente tabla

Tabla 11. *Definición de los métodos de inferencia más utilizados.*

Método de inferencia	Definición
Mamdani minimum inference, R_M	$\min(\mu, \mu_W(z)), \forall z$
Larsen product inference, R_L.	$\mu \times \mu_W(z), \forall z$
Drastic product inference, $R_{DP.}$	$\begin{cases} \mu & para\ \mu_W(z) = 1 \\ \mu_W(z) & para\ \mu = 1 \\ 0 & para\ \mu < 1\ y\ \mu_W(z) < 1 \end{cases}$
Bounded product inference, R_{BP}.	$\max(\mu + \mu_w(z) - 1, 0)$

Donde *μw* es la función de pertenencia del conjunto de salida w.

4.1.7 - Defusificación

La defusificación o *defuzzyfication* es un proceso matemático usado para convertir un conjunto borroso en un número real. El sistema de inferencia borrosa obtiene una conclusión a partir de la información de la entrada, pero es en términos difusos.

Existen diferentes métodos de defusificación que arrojan resultados distintos. Algunos de los más usados son el *Center of Area (COA), Middle of Maximum (MOM), Smallest of Maximum (SOM),* o *Largest of Maximum (LOM)*. A continuación se muestran ejemplos gráficos de cada método gracias a la herramienta *Fuzzy Logic Toolbox* de Matlab.

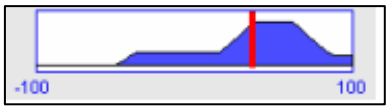

Figura 4-6. Defusificación con el método COA, *F=37.*

Figura 4-7. Defusificación con el método LOM, *F=60.*

Figura 4-8. Defusificación con el método SOM, *F=40.*

Figura 4-9. Defusificación con el método LOM, *F=50.*

4.2 - Control borroso vs PID

La teoría de conjuntos borrosos es usada en muchos campos técnicos como modelado, procesamiento de señales, sistemas expertos, etc., pero donde se usa con más frecuencia es en el campo de la ingeniería de control. ¿Por qué se usan estos tipos de controladores si los PID nos ofrecen resultados ya satisfactorios?

La mayor **ventaja** del control difuso es que provee una eficiente y efectiva metodología para desarrollar de forma experimental un controlador no lineal sin usar matemática avanzada. Esto significa que el controlador no necesita explícitamente el modelo del proceso a controlar.

Además, es posible trabajar tanto con sistemas SISO como MIMO, sin necesidad de utilizar matemática sofisticada. Esto es una gran ventaja si se tiene en cuenta la dificultad que tiene a veces el control de sistemas MIMO mediante PIDs.

Mediante simulaciones y prueba y error se puede llegar a crear un controlador difuso con una respuesta idónea. Para ello existen softwares como Matlab que ofrecen la posibilidad de desarrollar este tipo de controladores.

Su gran **desventaja** es que no cuenta con herramientas de análisis y diseño muy sofisticadas. Esto significa que es muy complicado encontrar un software matemático que genere la estructura del controlador, de ahí que el análisis de estabilidad o de cualquier otra propiedad sea muy complejo y solo está basado en que la implementación del conocimiento genere un sistema estable. Esto significa que la estabilidad no se garantiza desde el punto de vista matemático, aunque sí puede serlo desde el punto de vista experimental.

Otra desventaja de los controladores borrosos es que el ajuste de sus parámetros no cuenta con técnicas específicas. La cantidad de parámetros en un controlador difuso es mucho mayor que en un PID, ya que se puede variar el número y la forma de las funciones de pertenencia, añadir más o menos reglas, cambiar la relación entre ellas… añadir hay una gran variedad de definición de operadores, formas de conjuntos, etc.,

En aplicaciones donde se necesite mucha precisión (dada por la matemática del sistema) y no se pueda utilizar simulación, los controladores borrosos no tienen gran cabida.

4.3 - Diseño del controlador

Antes de empezar a diseñar el controlador, se deberá conocer el comportamiento del vehículo. Como ya se ha comentado con anterioridad, solamente se tendrán en cuenta las ecuaciones cinemáticas para simplificar el problema. Estas ecuaciones se muestran a continuación:

$$x_{K+1}=x_K-V\cdot\sin(\theta_K)\cdot h \qquad (4.2)$$

$$y_{K+1}=y_K+V\cdot\cos(\theta_K)\cdot h \qquad (4.3)$$

$$\theta_{K+1}=\theta_K+V\cdot\gamma_K\cdot h \qquad (4.4)$$

$$\gamma_{K+1}=\gamma_K+(1/T)\cdot(\gamma_{Rk}-\gamma_K)\cdot h \qquad (4.5)$$

Siendo x e y las coordenadas del vehículo, θ su orientación, V su velocidad, γ la curvatura que describen las ruedas del coche (curvatura de referencia), T la constante de tiempo y h el paso de integración. Se realizarán todas las pruebas con una constante de tiempo T=100 ms.

En la siguiente figura se observa un esquema del funcionamiento de éstas ecuaciones.

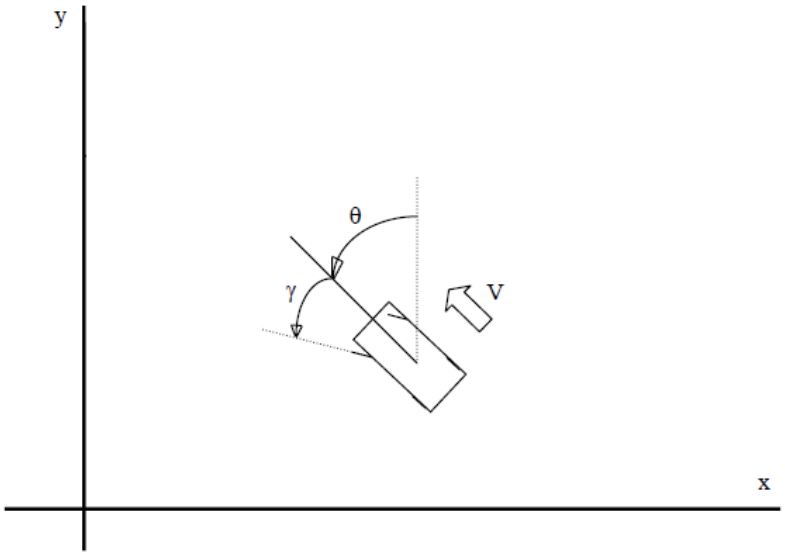

Figura 4-10. Esquema del funcionamiento del vehículo.

A continuación se muestra un esquema simplificado del sistema, donde se ve que las entradas al controlador serán la el error del robot en el eje x y el error de orientación, mientras que su salida será la curvatura de las ruedas y su posición y.

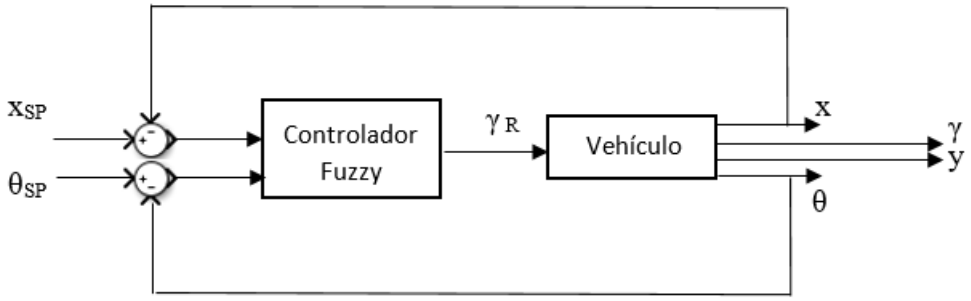

Figura 4-11. Esquema del controlador del vehículo móvil.

El controlador deberá hacer el seguimiento de rectas, las cuales vendrán dadas por el algoritmo RRT* previamente diseñado. Éstas rectas darán las entradas al controlador, ya que se podrá saber en todo momento el ángulo del vehículo deseado y su posición en eje *x* deseada.

Debido a que el controlador deberá seguir rectas, se harán pruebas para seguir una recta de 45º con diferentes controladores fuzzy. Una vez se diseñen, se probará el sistema con velocidades de v=10 m/s y v=30 m/s.

Para el diseño de los controladores borrosos se utilizará la herramienta de Matlab *"Fuzzy Logic Designer"*.

4.3.1 - Controlador 1

Para el primer controlador se hará una prueba inicial y se diseñará con 3 funciones de pertenencia para cada entrada y 3 funciones de pertenencia para su salida.

Tabla 12. *Funciones de pertenencia para error x.*

Nombre función	Tipo de función	Parámetros
Malo izquierda	Trapezoidal	a<-2; b=-2; c=-1; d=0
Buena	Triangular	a=-0.1; m=0; b=0.1
Malo derecha	Trapezoidal	a=0; b=1; c=2; d>2

Tabla 13. *Funciones de pertenencia para error ángulo.*

Nombre función	Tipo de función	Parámetros
Girado-derecha	Trapezoidal	a<-3.15; b=-3.15; c=-1.5; d=0
Buen ángulo	Triangular	a=-0.8; m=0; b=0.8
Girado-izquierda	Trapezoidal	a=0; b=1.5; c=3.15; d>3.15

Tabla 14. *Funciones de pertenencia para curvatura de referencia.*

Nombre función	Tipo de función	Parámetros
Giro derecha	Trapezoidal	a<-3; b=-3; c=-1.5; d=0
Continuar	Triangular	a=-0.6; m=0; b=0.6
Giro izquierda	Trapezoidal	a=0; b=1.5; c=3; d>3

Con la combinación de las entradas se obtienen 9 reglas con 3 diferentes salidas posibles. En la siguiente tabla se observa la salida del controlador con la combinación de las entradas:

Tabla 15. *Conjunto de reglas para el controlador 1.*

Reglas C1	Girado-derecha	Buen ángulo	Girado-izquierda
Malo izquierda	Giro-izquierda	Giro-derecha	Giro-derecha
Buena	Giro-izquierda	Continuar	Giro-derecha
Malo derecha	Giro-izquierda	Giro-izquierda	Giro-derecha

Una vez se tienen las reglas definidas se puede probar el controlador. Los resultados de estas pruebas se observan en las figuras 4-12 y 4-13, donde el vehículo intentará seguir la línea roja.

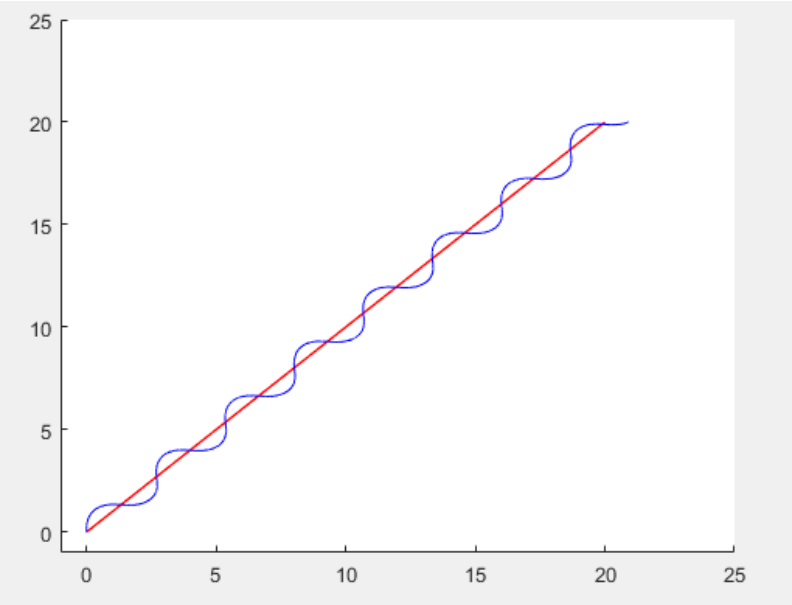

Figura 4-12. Comportamiento del vehículo (azul) con el controlador 1 y v=10m/s.

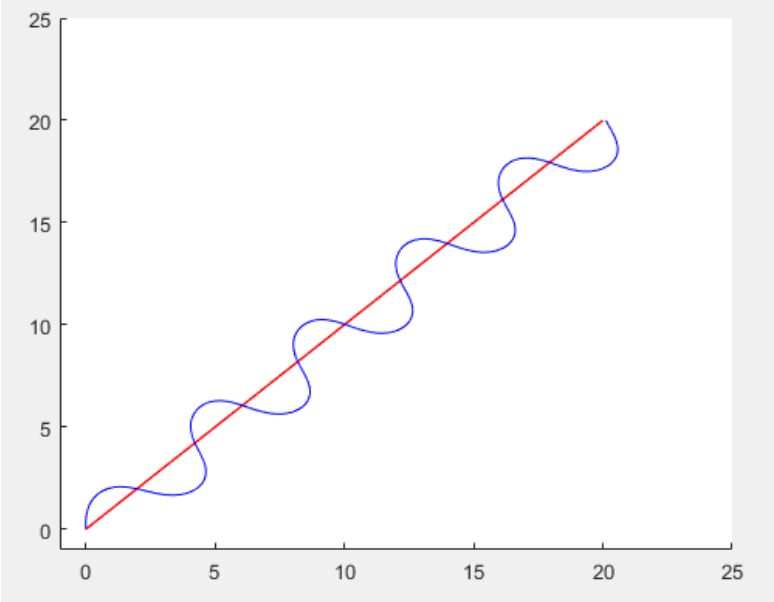

Figura 4-13. Comportamiento del vehículo (azul) con el controlador 1 y v=30m/s.

Observando las gráficas se puede decir que el controlador 1 es inestable y por lo tanto no será útil para nuestro sistema.

4.3.2 - Controlador 2

Se decide incrementar el número de funciones de pertenencia para la entrada del error del ángulo y para la salida del controlador. Las funciones para la entrada del error en el eje x serán las mismas que en el controlador 1.

Tabla 16. *Funciones de pertenencia para error ángulo.*

Nombre función	Tipo de función	Parámetros
Girado-derecha	Trapezoidal	a<-3.15; b=-3.15; c=-2.5; d=-1.5
Derecha-bueno	Triangular	a=-2; m=-1; b=0
Buen ángulo	Triangular	a=-0.4; m=0; b=0.4
Izquierda-bueno	Triangular	a=0; m=1; b=2
Girado-izquierda	Trapezoidal	a=1.5; b=2.5; c=3.15; d>3.15

Tabla 17. *Funciones de pertenencia para curvatura de referencia.*

Nombre función	Tipo de función	Parámetros
Giro derecha	Trapezoidal	a<-3; b=-3; c=-2.5; d=-1.5
Derecha recto	Triangular	a=-2; m=-1; b=0
Continuar	Triangular	a=-0.6; m=0; b=0.6
Izquierda recto	Triangular	a=0; m=1; b=2
Giro izquierda	Trapezoidal	a=1.5; b=2.5; c=3.15; d>3.15

Con la combinación de las entradas se obtienen 15 reglas con 5 diferentes salidas posibles, las cuales se observan a continuación:

Tabla 18. *Conjunto de reglas para el controlador 2.*

Reglas C2	Girado-derecha	Derecha-bueno	Buen ángulo	Izquierda-bueno	Girado-izquierda
Malo izquierda	Izquierda recto	Izquierda recto	Derecha recto	Giro derecha	Giro derecha
Buena	Giro izquierda	Izquierda recto	Continuar	Derecha recto	Giro derecha
Malo derecha	Giro izquierda	Giro izquierda	Izquierda recto	Derecha recto	Derecha recto

A continuación se muestran los resultados obtenidos con el controlador diseñado.

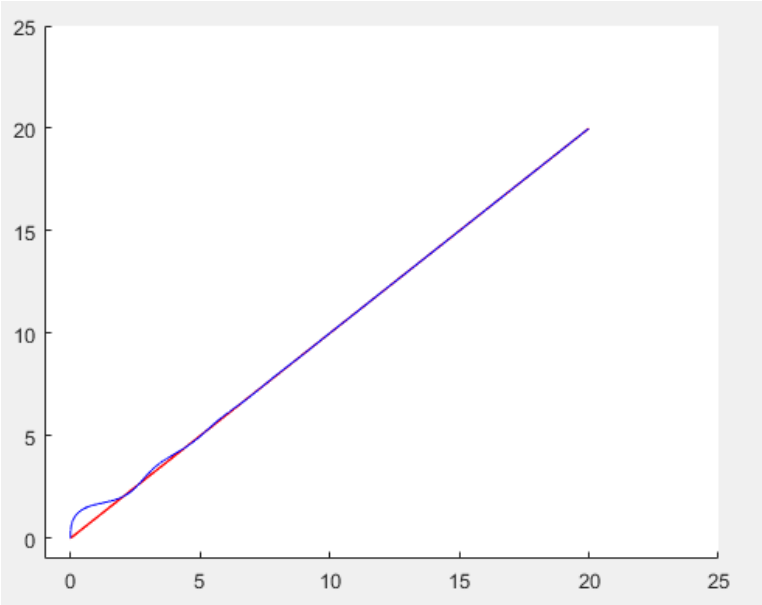

Figura 4-14. Comportamiento del vehículo (azul) con el controlador 2 y v=10m/s.

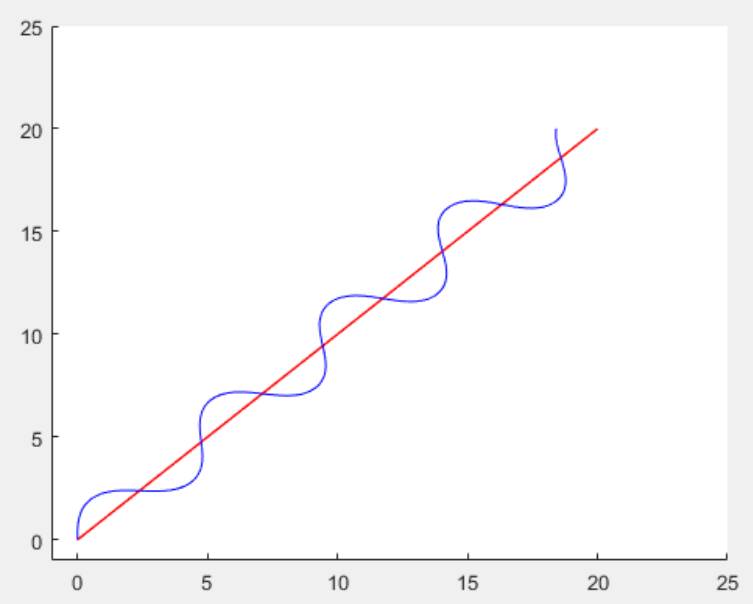

Figura 4-15. Comportamiento del vehículo (azul) con el controlador 2 y v=30m/s.

Esta vez se observa como el controlador tiene una respuesta rápida cuando la velocidad es de 10 m/s y además su error estacionario es prácticamente nulo. Sin embargo, cuando la velocidad es de 30 m/s el controlador sigue siendo inestable. En este caso, a partir de 20 m/s, el controlador será inestable.

4.3.3 - Controlador 3

Finalmente se decide diseñar un tercer controlador para intentar encontrar una respuesta aceptable para una velocidad de 30 m/s.

Se decide aumentar las funciones de pertenencia de la entrada error *x* y de la salida del controlador:

Tabla 19. *Funciones de pertenencia para error en x.*

Nombre función	Tipo de función	Parámetros
Malo izquierda	Trapezoidal	a<-5; b=-5; c=-2.5; d=-1
Izquierda Bueno	Triangular	a=-1.5; m=-0.75; b=0
Buena	Triangular	a=-0.6; m=0; b=0.6
Derecha bueno	Triangular	a=0; m=0.75; b=1.5
Malo derecha	Trapezoidal	a=1; b=2.5; c=5; d>5

Tabla 20. *Funciones de pertenencia para error ángulo.*

Nombre función	Tipo de función	Parámetros
Girado-derecha	Trapezoidal	a<-3.15; b=-3.15; c=-1.5; d=-0.75
Derecha-bueno	Triangular	a=-1; m=-0.5; b=0
Buen ángulo	Triangular	a=-0.5; m=0; b=0.5
Izquierda-bueno	Triangular	a=0; m=0.5; b=1
Girado-izquierda	Trapezoidal	a=0.75; b=1.5; c=3.15; d>3.15

Tabla 21. *Funciones de pertenencia para curvatura de referencia.*

Nombre función	Tipo de función	Parámetros
Todo derecha	Trapezoidal	a<-3; b=-3; c=-2.5; d=-1.5
Derecha	Triangular	a=-2; m=-1.25; b=-0.5
Derecha recto	Triangular	a=-0.8; m=-0.4; b=0
Continuar	Triangular	a=-0.05; m=0; b=0.05
Izquierda recto	Triangular	a=0; m=0.4; b=0.8
Izquierda	Triangular	a=0.5; m=1.25; b=2
Todo izquierda	Trapezoidal	a=1.5; b=2.5; c=3.15; d>3.15

Para este controlador se obtienen 25 reglas, combinando las 5 funciones de pertenencia de cada entrada. También se ha aumentado las funciones de pertenencia de salida, siendo esta vez de 7. Las reglas se muestran en la tabla 22:

Tabla 22. *Conjunto de reglas para el controlador 3.*

Reglas C3	Girado-derecha	Derecha-bueno	Buen ángulo	Izquierda-bueno	Girado-izquierda
Malo izquierda	Izquierda recto	Continuar	Derecha recto	Derecha	Derecha
Izquierda bueno	Izquierda	Continuar	Derecha recto	Todo derecha	Todo derecha
Buena	Todo izquierda	Todo izquierda	Continuar	Todo derecha	Todo derecha
Derecha bueno	Todo izquierda	Todo izquierda	Izquierda recto	Continuar	Derecha
Malo derecha	Izquierda	Izquierda	Izquierda recto	Continuar	Derecha recto

Los resultados obtenidos para el controlador 3 son los siguientes:

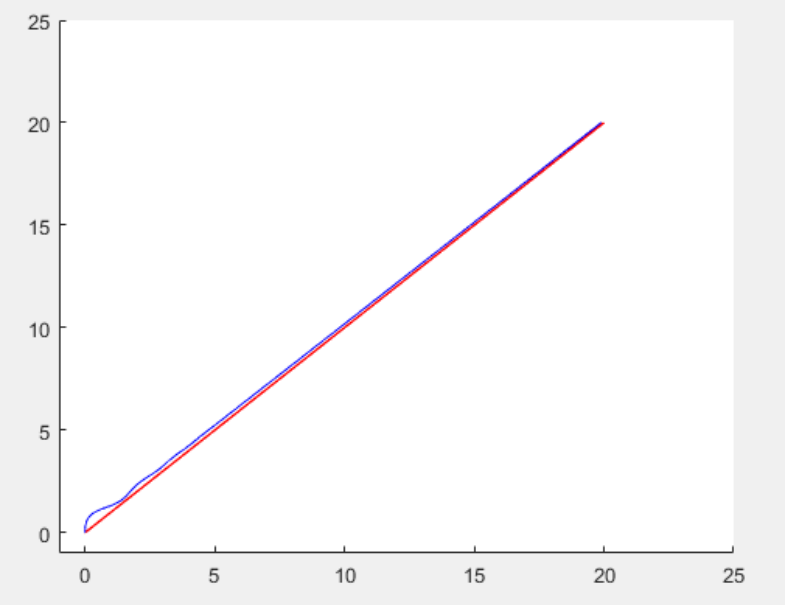

Figura 4-16. Comportamiento del vehículo (azul) con el controlador 3 y v=10m/s.

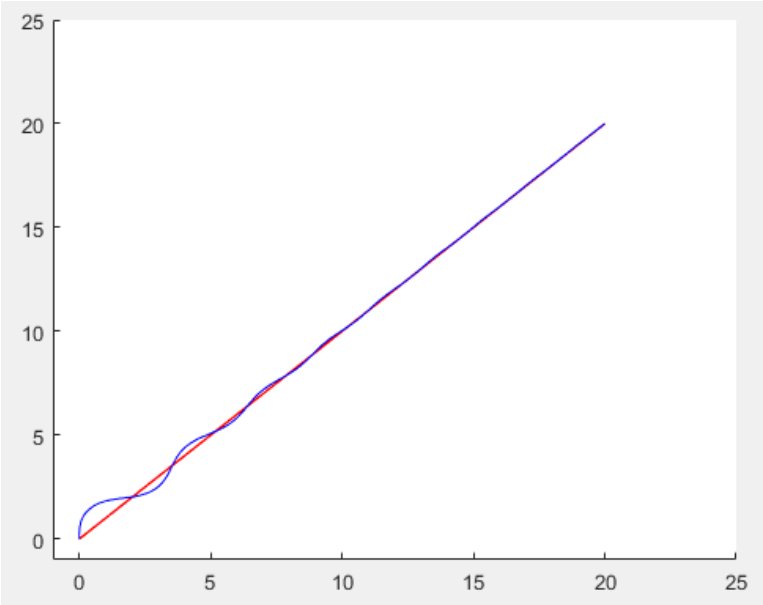

Figura 4-17. Comportamiento del vehículo (azul) con el controlador 2 y v=30m/s.

Con los resultados obtenidos se observa que esta vez el vehículo puede ser controlado a 30 m/s, mejorando así los otros controladores. Además, con este controlador el sistema no se vuelve inestable hasta llegar a los 40 m/s. Sin embargo, para una velocidad inferior, aunque este controlador muestre un sobre pico inferior al controlador 2, también se observa un error estático superior.

4.3.4 - Conclusiones

Observando el resultado de los tres controladores diseñados se puede concluir que la calidad de su respuesta será proporcional al número de reglas. Aun así, hay que tener cuidado con el diseño de éstas ya que si hay muchas reglas, éstas no se deben contradecir.

Con la comparación de los dos últimos controladores se ve que no hay uno mejor que otro. Si el vehículo funcionará con velocidades comprendidas 0 m/s y 25 m/s, el segundo controlador será el indicado. Sin embargo, si las velocidades son superiores a 25 m/s, se deberá usar el tercer controlador.

Como se ha comentado en el apartado 4.2, una de las desventajas de los controladores borrosos es el ajuste y la parametrización de este. Es decir, para obtener una respuesta específica, se puede hacer de muchas maneras: cambiarlas reglas, añadir reglas, añadir funciones de pertenencia, cambiar el tipo de función de pertenencia…Esto implica una

gran cantidad de tiempo ya que la mejor herramienta para observar si los cambios realizados son los deseados es mediante prueba y error.

Por el contrario, se puede obtener una respuesta muy buena sin la necesidad de utilizar ecuaciones diferenciales, transformadas de Laplace o funciones de transferencias.

5 - RESULTADOS

Una vez obtenido el sistema de control y el algoritmo de planificación de movimiento, se obtiene la planificación de la trayectoria, la cual, como se ha comentado en el apartado 2.1, ésta determina el movimiento del robot de una manera que respete sus limitaciones a lo largo de una solución de planificación de movimiento previamente establecida.

Para las últimas pruebas a realizar se utilizarán 3 circuitos diferentes y se cambiarán los parámetros para observar las diferencias entre los resultados obtenidos.

En estas pruebas se utilizará el algoritmo RRT*, ya que sin el post-procesado realizado en el apartado 3.3 el vehículo sería incapaz de seguir la ruta calculada.

Debido a la cantidad de parámetros que se pueden modificar, para las pruebas a realizar solamente se modificarán los siguientes:

- **α**
- **Velocidad**

Los parámetros fijos serán los siguientes:

- **ε=10**
- **Umbral=20**
- **Rq=200**

Se ha escogido esta parametrización fija ya que se ha demostrado con anterioridad un funcionamiento adecuado con estos valores.

A continuación se realizan las comparaciones con los diferentes circuitos y parámetros.

5.1 - Circuitos sin obstáculos

En la siguiente figura se muestra el circuito 1 y un ejemplo del cálculo que realiza el algoritmo para este circuito con los puntos de referencia especificados. En el siguiente circuito el punto inicial será [500 0]:

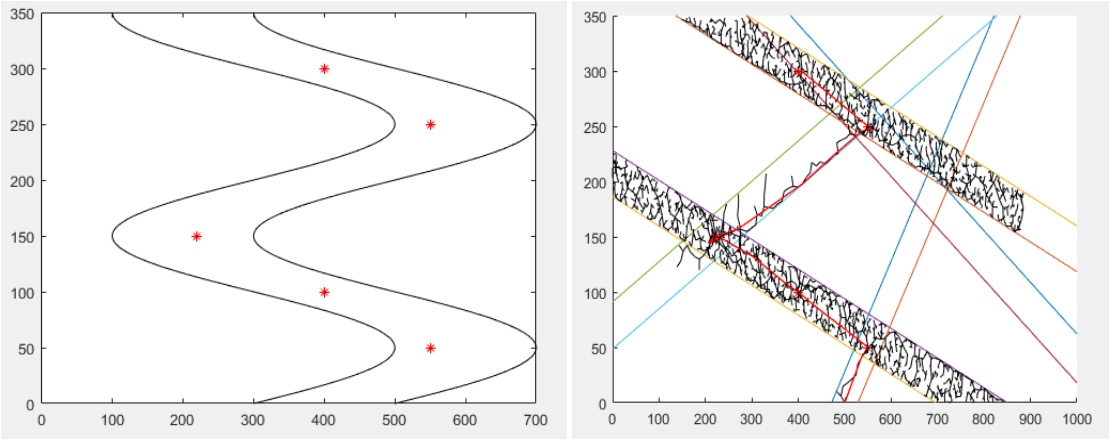

Figura 5-1. Circuito 1 (izquierda). Crecimiento del RRT* para el circuito 1 (derecha).

Para las siguientes pruebas se ha impuesto una velocidad de 10 m/s y por lo tanto se ha utilizado el controlador 2. En las siguientes figuras se muestra el resultado final del comportamiento del vehículo para α=1 (cálculo del coste sin tener en cuenta los giros) y para α=0 (cálculo del coste sin tener en cuenta la distancia).

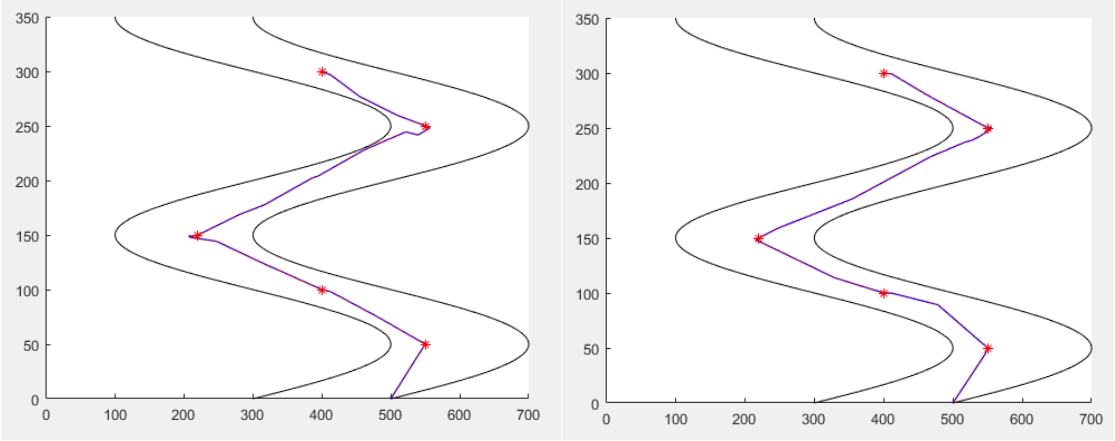

Figura 5-2. Comportamiento del vehículo en el primer circuito con α=1 (izquierda) y con α=0 (derecha).

En esta comparación se observa como la ruta especificada con α=0 es más fácil de seguir para un coche que con α=1. A simple vista el controlador puede seguir la ruta perfectamente en ambos casos. En las siguientes figuras se muestra una ampliación de los circuitos de la figura 5-2:

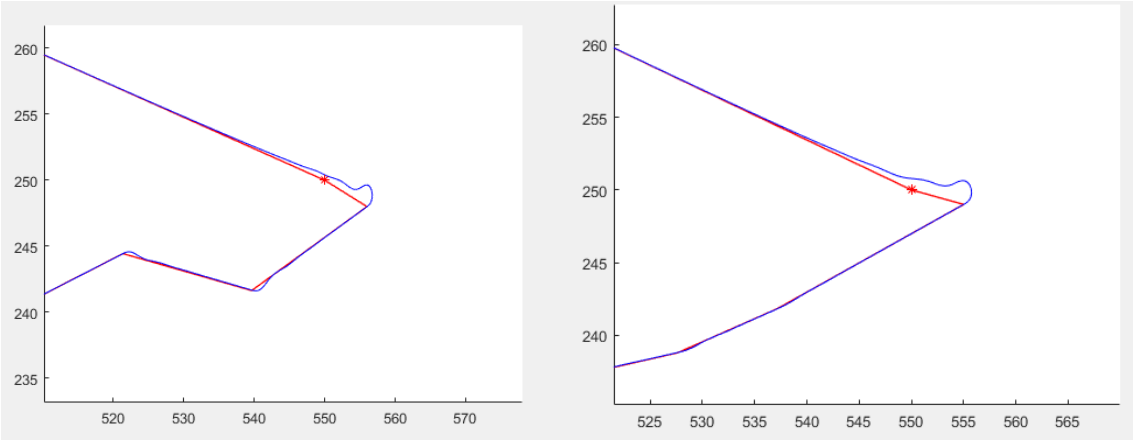

Figura 5-3. Comportamiento del controlador para un giro crítico con α=1 (izquierda) y con α=0 (derecha).

Teniendo en cuenta que las figuras anteriores contienen un punto crítico y que nuestro vehículo circula a 36 km/h, se puede concluir que el comportamiento es aceptable, ya que el giro que debe realizar es de 90º aproximadamente. El comportamiento general del coche se observa en la figura 5-4, donde se muestra la ruta a seguir sin cambios críticos.

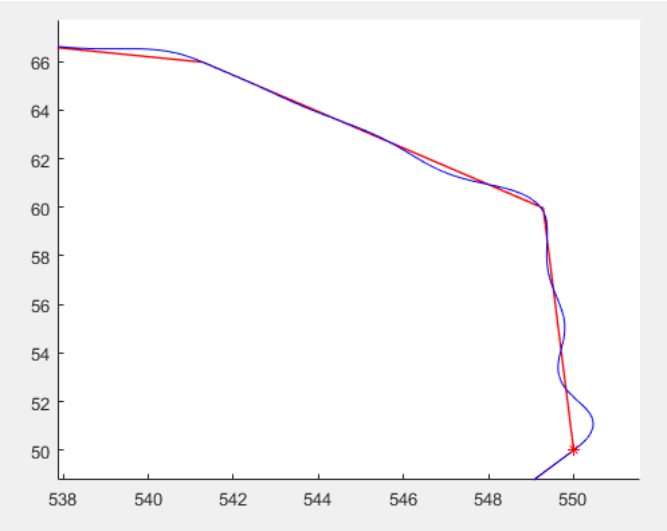

Figura 5-4. Comportamiento del vehículo móvil para V=10 m/s. Ruta calculada (rojo), vehículo (azul).

En las siguientes pruebas se ha decidido aumentar la velocidad hasta 40 m/s para estudiar el comportamiento del vehículo. Se decide observar si debido al incremento de velocidad el coche puede seguir la ruta planificada sin llegar a ningún conflicto. Por ello, como se ha visto en el apartado 4.3, también se ha cambiado del controlador 2 al 3 para que la respuesta del vehículo no sea inestable.

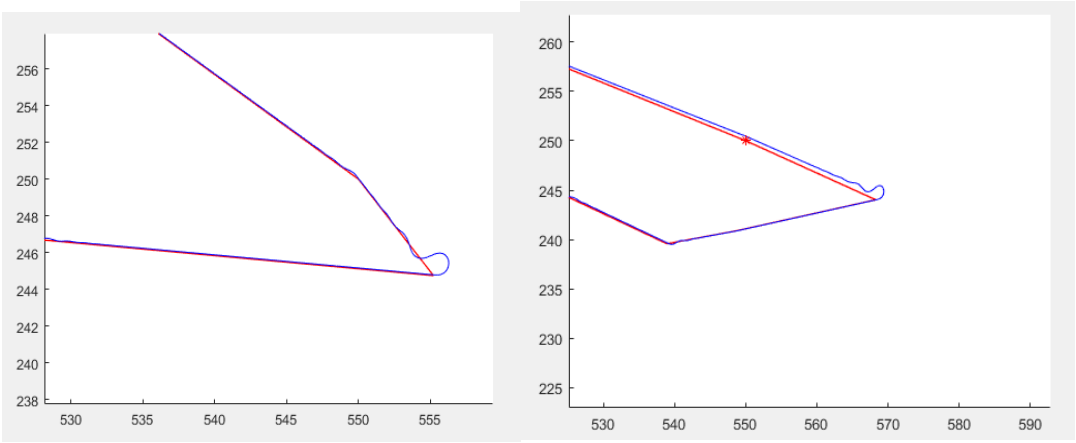

Figura 5-5. Comportamiento del controlador 3 para un giro crítico con una velocidad de 40 m/s (izquierda) y 10 m/s.

Estudiando los gráficos anteriores, se confirma el comportamiento esperado según lo comentado en el apartado 4.3.3. El controlador 3 obtiene un buen rendimiento para altas velocidades, pero para velocidades parecidas a las que el controlador 2 puede aceptar, éste contiene un error estático significativo.

Las siguientes pruebas se harán siempre con el controlador 2, con velocidades comprendidas entre 10 m/s y 20 m/s. Aunque el controlador 3 pueda aceptar mayores velocidades, se considera más inestable, ya que los sobrepicos que se muestran podrían ser difíciles de seguir según las medidas del vehículo y la ruta planificada.

En la siguiente figura se muestra un ejemplo del cálculo que realiza el algoritmo para el segundo circuito de ejemplo. En el siguiente circuito el punto inicial será [50 0] :

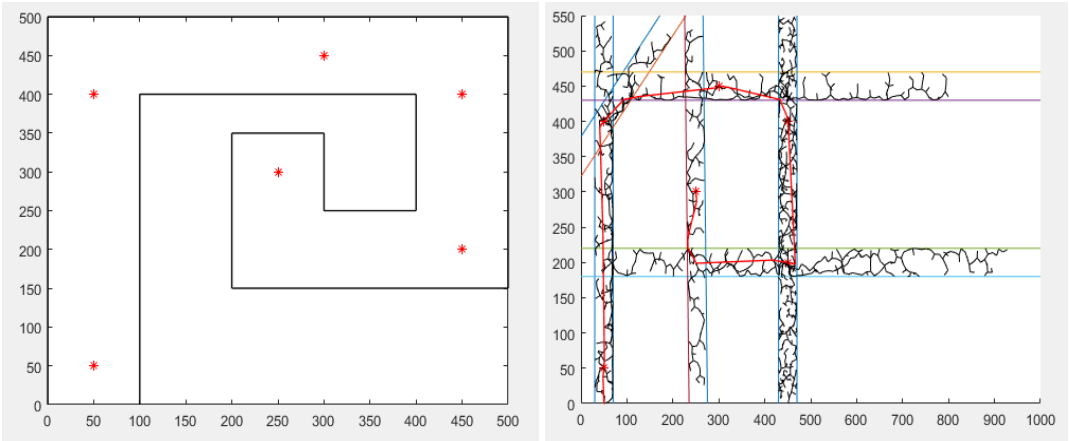

Figura 5-6. Circuito 2 (izquierda). Crecimiento del RRT* para el circuito 2 (derecha).

El siguiente circuito se ha realizado con α=0.998, por lo tanto el coste del algoritmo RRT* se ha calculado mediante la distancia y el ángulo de giro.

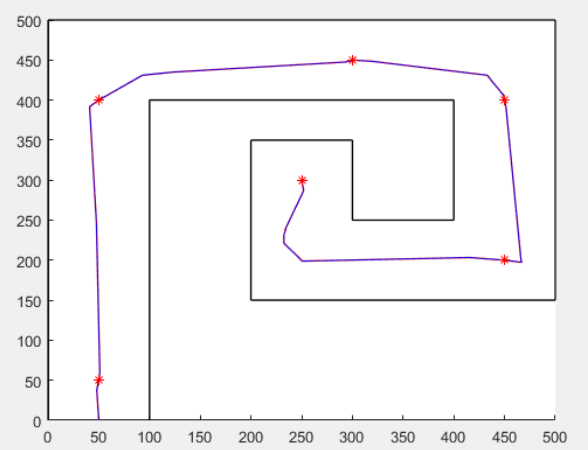

Figura 5-7. Comportamiento del vehículo para el circuito 2 con α=0.998.

En este ejemplo se observa con claridad como el algoritmo RRT* es capaz de encontrar una ruta óptima con solo 6 puntos de referencia, los cuáles están estratégicamente posicionados. Como se ha comentado con anterioridad, estos puntos deberían ser introducidos por otro algoritmo el cual no disponemos.

Seguidamente se muestran ejemplos del comportamiento del coche para este circuito.

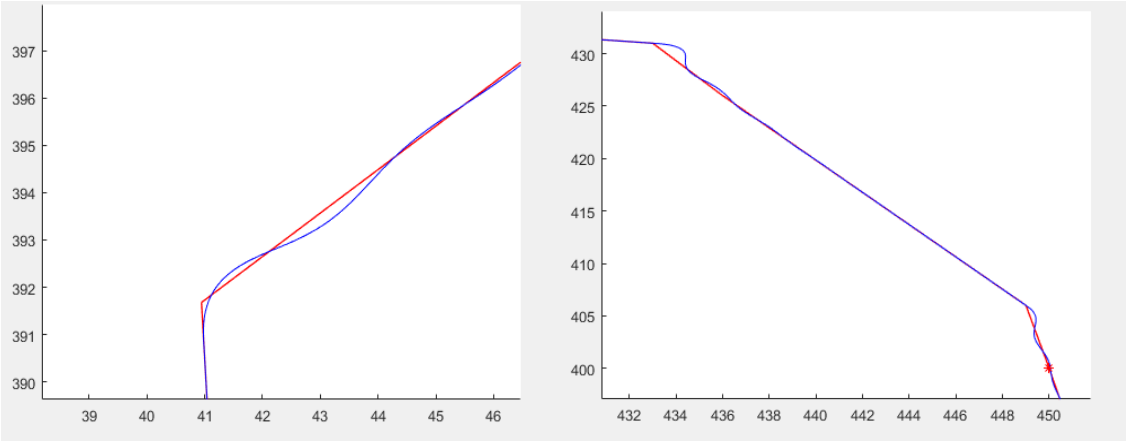

Figura 5-8. Comportamiento del vehículo móvil para V=10 m/s (izquierda) y V=20 m/s (derecha).

5.2 - Circuitos con obstáculos

Para finalizar, se vuelven a realizar pruebas, pero esta vez se añaden obstáculos en la carretera. En la siguiente figura se muestran las pruebas realizadas para el circuito 1 con α=0.998.

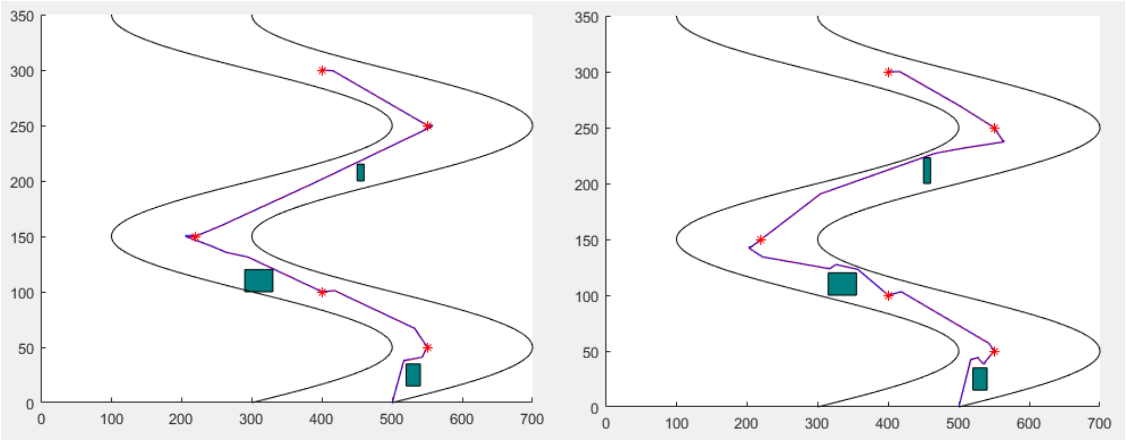

Figura 5-9. Comportamiento del vehículo en el primer circuito con diferente posicionamiento de los obstáculos.

Como se esperaba, al introducir obstáculos en la carretera, el RRT* los evita, llegando a dar una buena solución al problema con bastante rapidez. Si se restringe el umbral de búsqueda, el algoritmo tardará más en encontrar una solución, pero está será mejor. En la figura 5-10 se observa un ejemplo de esta búsqueda más restringida.

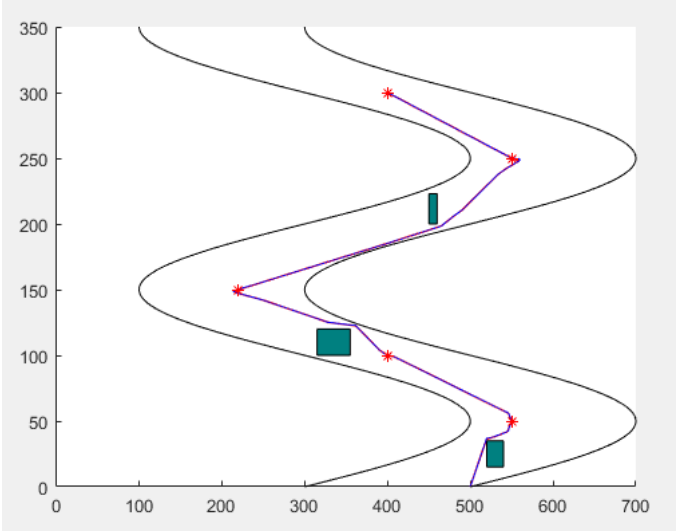

Figura 5-10. Comportamiento del vehículo en el primer circuito con obstáculos y búsqueda restrictiva.

Comparando con la figura 5-9, se puede comprobar como la última ruta calculada se acerca a la ruta óptima teniendo en cuenta el coste por distancia y por giros del vehículo. En las siguientes figuras se muestra el comportamiento del vehículo al pasar cerca de los obstáculos.

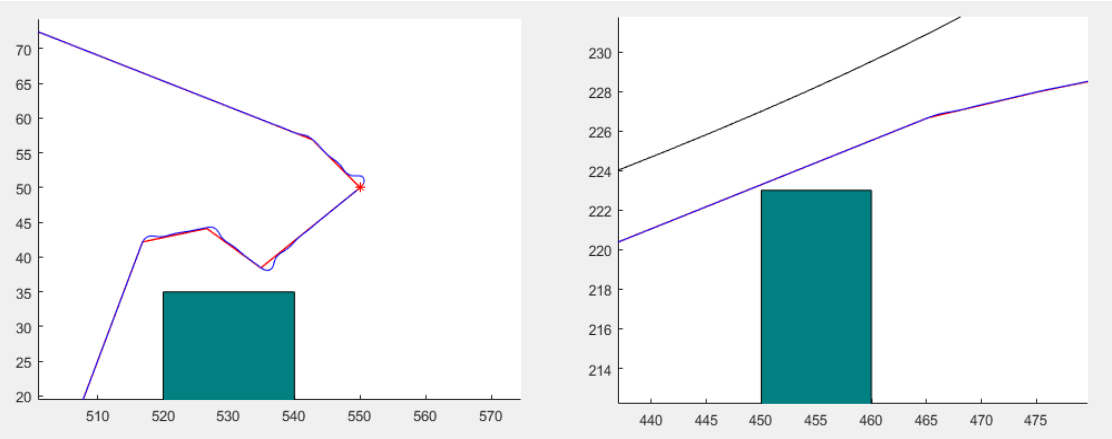

Figura 5-11. Ejemplo de comportamiento del vehículo al evitar los obstáculos.

Se observa como para ciertos tramos, el coche pasa muy cerca de los obstáculos. Para solucionar este problema simplemente se deberían conocer las medidas del vehículo y una vez conocidas, se podría dar un margen de seguridad a la hora de pasar cerca de los obstáculos. Para que esta solución fuera óptima, el control del vehículo también debería incluir la parte dinámica.

Para finalizar, se decide realizar una última prueba con un circuito simple, pero con un posicionamiento de los obstáculos muy restrictivo. En la siguiente figura se muestra un ejemplo del cálculo que realiza el algoritmo para el tercer circuito de ejemplo con α=0.998 y un punto inicial de [0 50] :

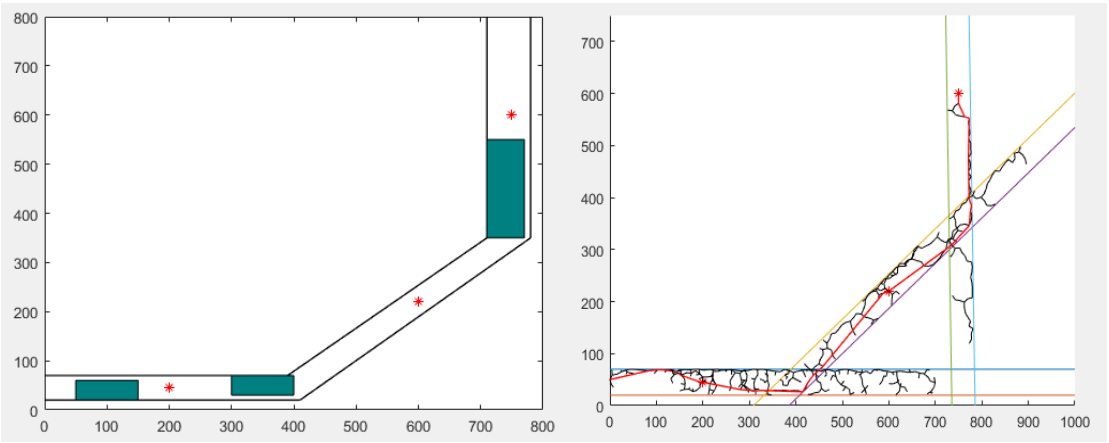

Figura 5-12. Circuito 3 (izquierda). Crecimiento del RRT* para el circuito 3 (derecha).

En la figura 5-13 se observa el comportamiento esperado, donde el algoritmo encuentra una ruta posible evitando todos los obstáculos.

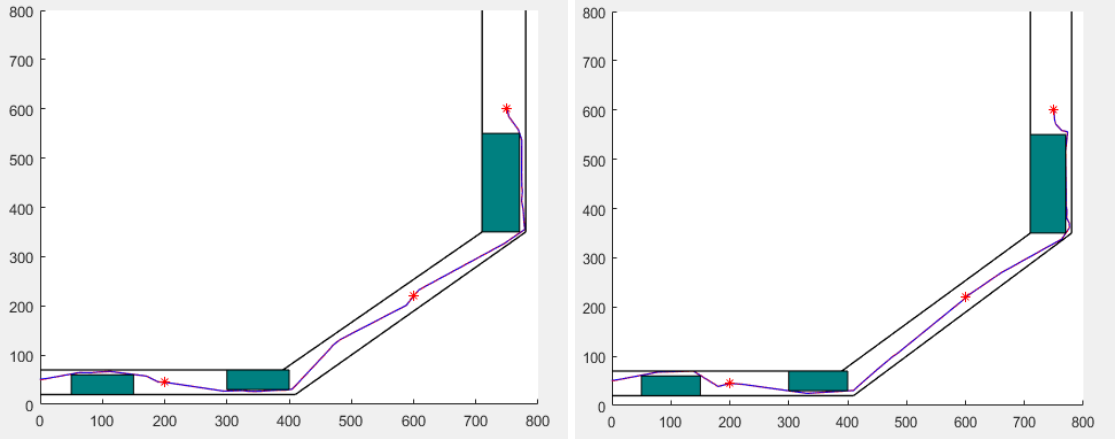

Figura 5-13. Comportamiento del vehículo en el tercer circuito con α=1 (izquierda) y con α=0 (derecha).

El posicionamiento de los obstáculos impide que las diferentes soluciones sean variadas, por eso, al cambiar el parámetro α no se aprecia gran diferencia, teniendo en cuenta que aún con los mismos parámetros, las soluciones que encuentre el RRT* siempre serán diferentes.

Por último, se hacen pruebas con diferentes velocidades para observar la respuesta del controlador cuando éste está cerca de los obstáculos.

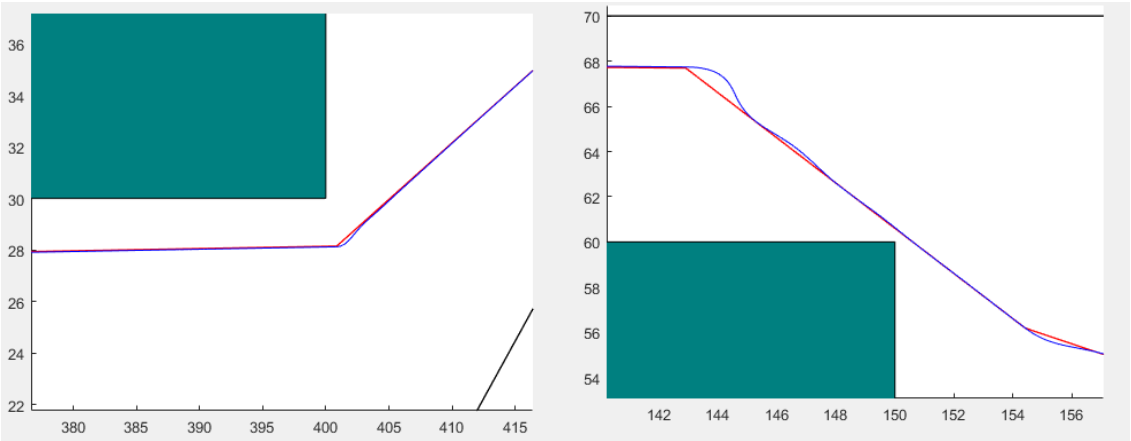

Figura 5-14. Comportamiento del vehículo al evitar los obstáculos con V=10 m/s (izquierda) y con 20 m/s (derecha).

Como era de esperar, al incrementar la velocidad, la ruta especificada es más difícil de seguir y por consiguiente, el sobrepico del controlador también aumenta. Sin embargo, como se ha visto con anterioridad, nuestro vehículo puede ser controlado sin problemas a 20 m/s.

Concluyendo, los resultados obtenidos son muy positivos ya que muestran el comportamiento que se deseaba. Con sólo introducirle al sistema los puntos de paso éste es capaz de mover el coche del punto inicial al punto final con la máxima precisión posible. También se introduce la posición de los obstáculos, aunque esto se podría evitar si no se trabajase en simulación y el vehículo tuviese un sistema de visión artificial. Aun teniendo que saber la posición de los obstáculos, acabamos obteniendo un vehículo completamente autónomo.

6 - IMPACTO MEDIOAMBIENTAL

El estudio de los algoritmos de planificación de movimiento se basa principalmente en el intento de automatización de tareas de búsqueda. Estas tareas suelen asociarse a robots móviles que se encargan de ejecutar una acción específica. En nuestro caso particular, el algoritmo estudiado se ha planteado incluirlo en un coche o vehículo móvil de manera que este pueda funcionar autónomamente.

Estos algoritmos podrían afectar al medio ambiente positivamente de manera indirecta. La futura y posible autonomía de los vehículos implicará que estos conocerán la ruta a seguir con anterioridad. De esta manera, al no ser interferidos por humanos, los vehículos podrán optimizar el uso del combustible y así contaminar lo menos posible.

7 - PRESUPUESTO

El presupuesto de este trabajo se basa principalmente en las horas que ha realizado el autor para poder finalizarlo. No se ha implementado en ningún vehículo real por lo que solamente se computará como gasto las licencias de los principales softwares utilizados.

Tabla 23. *Presupuesto del trabajo final.*

Concepto	Tiempo	Precio	Coste
Estudio de los algoritmos de planificación	40h	-	-
Diseño, cálculo del algoritmo e implementación	280h	35€/h	9800 €
Redacción	120h	15€/h	1800 €
Microsoft Office	6 meses	7€/mes	42 €
Matlab	6 meses	800€/año	400 €
Beneficio (15%)	-	-	1806,3 €
IVA (21%)	-	-	2908,1 €

Con la tabla 23, se concluye que el presupuesto total del trabajo realizado es de **16756,4€**.

Como se observa, la mayor parte del costo proviene de las horas realizadas diseñando y programando el algoritmo de planificación.

8 - CONCLUSIONES

En el siguiente apartado se explicarán las principales conclusiones extraídas sobre el presente trabajo.

Previamente a la programación del algoritmo de planificación de trayectoria se ha tenido que hacer un estudio sobre los algoritmos de planificación de movimiento. Como se ha comentado previamente, es importante diferenciar estos dos aspectos. La planificación del movimiento ignora las limitaciones y restricciones y se centra en las traslaciones y rotaciones requeridas para mover el objeto o robot móvil. Por el contrario, la planificación de la trayectoria incluye el problema de determinar el movimiento del robot de una manera que respete sus limitaciones a lo largo de una solución de planificación de movimiento previamente establecida.

Para obtener un buen algoritmo de planificación de trayectoria, se ha tenido que escoger un correcto algoritmo de planificación de movimiento, teniendo en cuenta las restricciones de nuestro vehículo y las características del sistema. Dentro del gran abanico de posibles algoritmos, se ha decidido utilizar los algoritmos que comprenden métodos estocásticos. Esto es debido a las ventajas que ofrecen respecto a los métodos *roadmap*, siendo los primeros sencillos de implementar y con una carga computacional menor.

Entre los posibles algoritmos de movimiento basados en métodos estocásticos se ha hecho hincapié en los RRT y los PRM. Observando su comportamiento, se ha visto que los PRMs encuentran una solución con mayor facilidad que los RRT, sin embargo, la carga computacional de los PRMs aumenta exponencialmente cuánto más complejo sea el sistema. Otra desventaja de los PRMs (en este caso crucial para el problema a tratar), es la poca flexibilidad del algoritmo, la cual hace que sea difícil de implementar en robots no holonómicos. De esta manera, se ha decidido utilizar los algoritmos RRT, concretamente el RRT estrella. Una de las ventajas de los RRTs es su fácil modificación. Se ha explotado esta ventaja utilizando el algoritmo RRT*, el cual optimiza la ruta si se compara con el RRT original.

Aun obteniendo un algoritmo de planificación el cual ofrezca una solución a nuestro problema, éste debe modificarse para que se adapte a las necesidades del vehículo móvil. Por ende, se han realizado una serie de modificaciones con la intención de que la ruta calculada pueda ser seguida por un coche real.

De todas las modificaciones hechas para el procesado del algoritmo, cabe destacar el agregado del parámetro α. Gracias a éste parámetro se ha podido llegar a una buena solución evitando giros imposibles de realizar por un robot no holonómico. Junto a ésta modificación, se ha añadido la variable del ángulo (innecesaria para el RRT pero indispensable para el control móvil), la división del problema mediante múltiples puntos de referencia y la simplificación geométrica de la búsqueda de estos puntos.

Al finalizar la programación del RRT estrella, se ha diseñado el controlador del vehículo. La principal peculiaridad del control del coche es que éste se ha basado en la lógica difusa, la cual se basa en las ambigüedades con las que se percibe una variable. Estas ambigüedades son modeladas mediante los conjuntos borrosos, los cuales son la base para la lógica difusa. De esta manera, se intenta emular el pensamiento humano.

Los controladores difusos usan reglas que combinan uno o más conjuntos borrosos de entrada y le asocian un conjunto borroso de salida. A estas reglas se les llama reglas borrosas o *fuzzy rules*. Son afirmaciones del tipo SI-ENTONCES y asocian los conjuntos borrosos de entrada mediante operaciones lógicas borrosas AND, OR, etc. En nuestro caso particular, un ejemplo de regla borrosa utilizada sería: “SI el coche está a la izquierda (respecto al eje x) de su consigna y también está orientado a la izquierda – ENTONCES gira las ruedas totalmente hacia la derecha”.

Mediante la implementación de los conjuntos borrosos y todas las reglas en el *Fuzzy Logic Toolbox* de Matlab nos ha permitido crear un buen control sin necesidad de cálculos avanzados. Así, utilizando técnicas de inteligencia artificial, se consigue un control cinemático del vehículo con resultados muy positivos, obteniendo un comportamiento muy parecido al que se podría esperar con un PID.

Finalmente, se prueba el algoritmo con el control diseñado y se observa un comportamiento esperado, siendo el coche capaz de evitar los obstáculos y llegar al punto final pasando por la carretera diseñada. Mediante técnicas de inteligencia artificial se consigue solucionar el problema planteado inicialmente. Este problema ofrece una gran complejidad e información incompleta, como la disposición exacta de la carretera. Sin embargo, mediante algoritmos relativamente sencillos como puede ser el crecimiento de un árbol, se consigue un buen resultado. Se puede afirmar que se ha resuelto un problema mediante ***Soft computing***.

Soft Computing es una rama de la Inteligencia Artificial que engloba diversas técnicas empleadas para solucionar problemas que manejan información incompleta, con incertidumbre y/o inexacta. Estos problemas serían muy complicados de solucionar con métodos analíticos y convencionales. Por el contrario, al simplificar el problema y utilizar otros métodos aparentemente más sencillos, conseguimos llegar a solucionar el problema.

Futuras implementaciones

De la manera que presentamos éste algoritmo, sería inviable de implementarlo en un vehículo que estuviera rodeado de otros vehículos o personas. Sin embargo, como se ha visto en el presente trabajo, los algoritmos RRT nos ofrecen una gran variedad de modificaciones. Con los sensores que tuviera el coche, se podría hacer que el RRT* se actualizase al observar un cambio en los obstáculos. Así, el árbol estaría en cambio constante y evitaría que el coche colisionase con obstáculos variables (personas, coches…).

De la misma manera, se debería realizar un control más completo. Se podría seguir utilizando un controlador con lógica difusa, pero se debería tener en cuenta las características dinámicas del coche y de la carretera.

Para finalizar, es obvio que solamente con un buen algoritmo de planificación de trayectorias no se podría incluir éste coche en la carretera. Para ello se necesitaría un nivel computacional mucho más elevado, incluyendo normas de tráfico, un sistema GPS, sensores de alta gama, sistema de comunicaciones, etc.

9 - REFERENCIAS

[1] .J.C. Latombe. (1991). *Robot Motion Planning.* Kluwer Academic Publishers, Boston, MA.

[2] Aurenhammer, F. (1991) *Voronoi diagrams: A survey of fundamental geometric data structure*. ACM Comput. Surv.,nº 23.

[3] Amato, N.M., Yan Wu. (1996) *A randomized roadmap method for path and manipulation planning*.

[4] Kavraki, L.E., Kolountzakis, M.N., Latombe, J-C. (1996a). *Analysis of Probabilistic roadmaps for path planning* IEEE Int"l Conf. on Robotics and Automation.

[5] Steven M. LaValle. (2006) *Planning Algorithms*. Cambridge University Press.

[6] Diego Antonio López García (2012), *Nuevas aportaciones en algoritmos de planificación para la ejecución de maniobras en robots autónomos no holónomos,* (Tesis doctoral). Universidad de Huelva, Huelva.

[7] Fortune, S. Wilfong, G. (1988). *Planning Constrained Motion*. Proccedings of the Fourth ACM Symposium on Computation Geometry, 440-459.

[8] LaValle, S.M. (1998) *Rapidly-exploring random trees: A new tool for path planning*. Computer Science Dept., Iowa State University.

[9] Julio César Regalado. (1999). *Lógica difusa aplicada al control de voltaje y potencia reactiva en subestaciones de distribución.* Universidad de Piura, Perú.

[10] Mathworks Inc (2017). What is Fuzzy Logic? [Fecha de consulta: 3 de abril de 2018] Disponible en: https://www.mathworks.com/help/fuzzy/what-is-fuzzy-logic.html

[11] Wikipedia (2013). Rapidly-exploring Random tree [Fecha de consulta: 28 de noviembre de 2017] Disponible en: https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree

[12] Mathworks Inc (2016). Probabilistic Roadmaps (PRM). [Fecha de consulta: 5 de mayo de 2018] Disponible en: https://es.mathworks.com/help/robotics/ug/probabilistic-roadmaps-prm.html

[12] Georgia College of Tech Computing (2011). PRM vs RRT vs RRT*. [Fecha de consulta: 28 de abril de 2018] Disponible en: https://www.cc.gatech.edu/~dellaert/11SAI/Topics/Entries/2011/2/21_PRM,_RRT,_and_RRT_.html

[13] Wikipedia (2017). Soft Computing. [Fecha de consulta: 1 de junio de 2018] Disponible en: https://es.wikipedia.org/wiki/Soft_Computing

ANEXOS

En el apartado de anexos se mostrará el código realizado para la programación del algoritmo de planificación de trayectorias.

Este código se dividirá en dos partes: Código principal y Funciones. En el código principal está la realización del RRT, la implementación del control y todas las llamadas a las diferentes funciones. En el apartado de funciones se realizan los diferentes cálculos repetitivos como el cálculo de la distancia entre nodos, el error en x del controlador, la diferencia de ángulo entre nodos…

1- CÓDIGO PRINCIPAL

```
% RRT* algorithm with collision avoidance.
%
% nodes:    Contiene la lista de los nodos explorados. Cada nodo
contiene sus coordenadas,
%           su coste y su respectivo padre.
%
% Breve descripción del algoritmo:
% 1. Se escoge un nodo aleatorio q_rand.
% 2. Se encuentra el nodo mas cercano a q_rand (q_near)
% 3. Se encuentra q_new. Comprueba que no haya obstáculo.
% 4. Se actualiza el coste de llegada hasta q_new. El nodo más cercano
%    es el padre de q_new
% 5. Se busca un nodo con menor coste hacia q_new con un radio
determinado
% (q_neares)
% 6. De todos los nodos q_nearest, selecciona el nodo con menor coste
hacia
%     q_new siempre que no haya obstáculo. Se actualiza el padre de
q_new.
% 7. Se añade q_new a la lista de nodos
% 8. Continúa hasta que se alcanza q_goal.

clearvars
clear all
close all
global coordx coordy
%%%%%CONTROLADOR%%%%%%

gamma=0;
V=10;
T=0.1;
Ts=0.1;
hh=0.005;
pract2=readfis('pruebas');
referencia=0;
Ttheta=0;

%%%%%%%%%%%%%%
x=0;
x_max = 1000;
y_max = 550;
```

```
x_min = 0;
y_min = 0;
%obstacle = [50,0,500,100];
%obstacle1= [0,120,800,50];
EPS = 10;
umbral_angulo=25;
umbral_goal=20;
threshold=20;  %ancho del rectángulo donde crece el árbol
threshold2=threshold;
r = 200;        % Dentro de un radio r, encuentra todos los nodos
existentes
dist_max=500;
ParamCost=0;

new_umbral=false;
q_rand=0;
change=0;




q_start.cost = 0;
q_start.parent = 0;
posicion=1;
continuar=true;

coordx  = [550 400 220 550 400 ];
coordy  = [50 100 150 250 300];
ptheta = [45 165 20 155 165];
q_start.coord = [500 0];

% coordx  = [50 50 300 450 450 250];
% coordy  = [50 400 450 400 200 300 ];
% ptheta = [90 45 0 270 180 91];
% q_start.coord = [50 0];

for i=1:length(coordx)
    q_goal(i).coord(1)=coordx(i);
    q_goal(i).coord(2)=coordy(i);
end

node_goal=q_goal;

for i=1:length(coordx)
    q_goal(i).theta=ptheta(i);
end

figure(1)
axis([x_min x_max y_min y_max])
%rectangle('Position',obstacle,'FaceColor',[0 .5 .5])
%rectangle('Position',obstacle1,'FaceColor',[0 .5 .5])
hold on


for h=1:1:length(q_goal)

carretera(q_goal(h).coord,q_goal(h).theta,threshold)
```

```
end


%Se utiliza variable nodes para que cuando encuentre q_goal(n), la
búsqueda
%de q_goal(n+1) empiece desde q_goal(n)

for aux = 1:1:length(q_goal)
    if aux==1
        nodes(1) = q_start;
    else

    nodes(1) = node_goal(aux-1);
    nodes(1).cost=0;
    nodes(1).parent=0;
    end
    acabar=false;

     % Se acaba si el nodo final ha sido encontrado
while acabar==false
    actualizar=false;

     q_rand = [floor(rand(1)*x_max) floor(rand(1)*y_max)];

    for j = 1:1:length(nodes)

        %Observa si se llega a q_goal con el ángulo deseado

        if ((nodes(j).coord(1)+umbral_goal > q_goal(aux).coord(1)) &
(nodes(j).coord(1)-umbral_goal < q_goal(aux).coord(1))) &
((nodes(j).coord(2)+umbral_goal > q_goal(aux).coord(2)) &
(nodes(j).coord(2)-umbral_goal < q_goal(aux).coord(2)))

            %Si el nuevo nodo coincide con q_goal, se calcula el
ángulo con
            %el nodo anterior
            if (q_goal(aux).coord(1)==nodes(j).coord(1)) &&
(q_goal(aux).coord(2)==nodes(j).coord(2))
                coord_x= q_goal(aux).coord(1)-nodes(j-1).coord(1);
                coord_y= q_goal(aux).coord(2)-nodes(j-1).coord(2);
                angulo=angle(coord_y,coord_x);
                node_goal(aux).cost = dist(nodes(j-1).coord,
q_goal(aux).coord) + nodes(j-1).cost;
            else
                coord_x= q_goal(aux).coord(1)-nodes(j).coord(1);
                coord_y= q_goal(aux).coord(2)-nodes(j).coord(2);
                angulo=angle(coord_y,coord_x);
                node_goal(aux).cost = dist(nodes(j).coord,
q_goal(aux).coord) + nodes(j).cost;
            end

      %Comprueba que se llegue al destino con la orientación deseada

             if angulo+umbral_angulo>360

                if  (angulo+umbral_angulo-360>q_goal(aux).theta) ||
(angulo+umbral_angulo>q_goal(aux).theta && angulo-
umbral_angulo<q_goal(aux).theta)
                    q_final=nodes(j);
```

```
                    padre_goal=j;
                    acabar=true;
                end

            elseif (angulo-umbral_angulo<0)

                if  (angulo-umbral_angulo+360<q_goal(aux).theta) ||
(angulo+umbral_angulo>q_goal(aux).theta && angulo-
umbral_angulo<q_goal(aux).theta)
                    q_final=nodes(j);
                    padre_goal=j;
                    acabar=true;
                end

              elseif angulo+umbral_angulo>q_goal(aux).theta &&
angulo-umbral_angulo<q_goal(aux).theta
                q_final=nodes(j);
                padre_goal=j;
                acabar=true;
             end

        end
    end

    % if acabar==true
    %    break
    % end

    % Se escoje el nodo más cercano de entre los existentes
    ndist = [];
    for j = 1:1:length(nodes)
        n = nodes(j);
        tmp = dist(n.coord, q_rand);
        %Se actualiza ndist
        ndist = [ndist tmp];
    end
    [val, idx] = min(ndist);
    q_near = nodes(idx);

    %Se encuentra q_new
    q_new.coord = steer(q_rand, q_near.coord, val, EPS);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%CALCUL
O DE Q_NEW Y COLISION%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if aux>1

        if noCollision2(q_goal(aux-
1).coord,q_goal(aux).coord,q_goal(aux).theta,threshold)

            continuar=true;
        else

            continuar=false;
        end


    end
```

```
    if continuar==true
         if
noCollision2(q_new.coord,q_goal(aux).coord,q_goal(aux).theta,threshold
) && dist(q_new.coord,q_goal(aux).coord)<dist_max
            colision=false;
         else
             colision=true;
         end

    elseif continuar==false && new_umbral==false

        if noCollision2(q_new.coord,q_goal(aux-1).coord,q_goal(aux-
1).theta,threshold) &&
noCollision2(q_new.coord,q_goal(aux).coord,q_goal(aux).theta,threshold
)
            colision=false;
            new_umbral=true;

        elseif noCollision2(q_new.coord,q_goal(aux-
1).coord,q_goal(aux-1).theta,threshold) &&
dist(q_new.coord,q_goal(aux).coord)<dist_max
             colision=false;

        elseif (q_goal(aux).theta==q_goal(aux-1).theta)
                threshold=threshold+1;
                colision=true;
        else

            colision=true;
        end

    elseif continuar==false

        if
noCollision2(q_new.coord,q_goal(aux).coord,q_goal(aux).theta,threshold
) && dist(q_new.coord,q_goal(aux).coord)<dist_max
            colision=false;
        else
            colision=true;
        end
    end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%INSERCION DE
Q_NEW%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      if  colision==false
        %Se dibuja el árbol
        figure(1)
        line([q_near.coord(1), q_new.coord(1)], [q_near.coord(2),
q_new.coord(2)], 'Color', 'k', 'LineWidth', 0.2);
        drawnow
        hold on

        if (q_near.parent==0) && (aux==1)
            Costeang=0;

        elseif (q_near.parent==0) && (aux>1)
           Angulo1=angle(q_new.coord(2)-
q_near.coord(2),q_new.coord(1)-q_near.coord(1));
           Angulo2=Angulo_Llegada;
```

```
           Costeang=(abs(Angulo1-Angulo2))^2;
        else
            Angulo1=angle(q_new.coord(2)-
q_near.coord(2),q_new.coord(1)-q_near.coord(1));
            Angulo2=angle(q_near.coord(2)-
nodes(q_near.parent).coord(2),q_near.coord(1)-
nodes(q_near.parent).coord(1));
            Costeang=(abs(Angulo1-Angulo2))^2;
        end

        if Costeang>32400

            if Angulo1>Angulo2
               Costeang=(360+Angulo2-Angulo1)^2;
            else
               Costeang=(360+Angulo1-Angulo2)^2;
            end
        end

        q_new.cost = dist(q_new.coord,
q_near.coord)*ParamCost+Costeang*(1-ParamCost)+ q_near.cost;

        q_nearest = [];
        neighbor_count = 1;
        for j = 1:1:length(nodes)
            if  dist(nodes(j).coord, q_new.coord) <= r
                q_nearest(neighbor_count).coord = nodes(j).coord;
                q_nearest(neighbor_count).cost = nodes(j).cost;
                q_nearest(neighbor_count).parent = nodes(j).parent;
                neighbor_count = neighbor_count+1;
            end
        end

        % Initialize cost to currently known value
        q_min = q_near;
        C_min = q_new.cost;

        % Iterate through all nearest neighbors to find alternate
lower
        % cost paths

        for k = 1:1:length(q_nearest)

            if (q_nearest(k).parent==0) && (aux==1)
                Costeang=0;
            elseif (q_nearest(k).parent==0) && (aux>1)
               Angulo1=angle(q_new.coord(2)-
q_nearest(k).coord(2),q_new.coord(1)-q_nearest(k).coord(1));
               Angulo2=Angulo_Llegada;
               Costeang=(abs(Angulo1-Angulo2))^2;
            else
                Angulo1=angle(q_new.coord(2)-
q_nearest(k).coord(2),q_new.coord(1)-q_nearest(k).coord(1));
                Angulo2=angle(q_nearest(k).coord(2)-
nodes(q_nearest(k).parent).coord(2),q_nearest(k).coord(1)-
nodes(q_nearest(k).parent).coord(1));
                Costeang=(abs(Angulo1-Angulo2))^2;

            end
```

```
            if Costeang>32400      % (180^2)
               if Angulo1>Angulo2
                    Costeang=(360+Angulo2-Angulo1)^2;
               else
                    Costeang=(360+Angulo1-Angulo2)^2;
               end
            end

         if continuar==false && new_umbral==false

           if noCollision2(q_new.coord,q_goal(aux-
1).coord,q_goal(aux-1).theta,threshold) && (q_nearest(k).cost +
dist(q_nearest(k).coord, q_new.coord)*ParamCost+Costeang*(1-
ParamCost))<= C_min
               q_min = q_nearest(k);
               C_min = q_nearest(k).cost + dist(q_nearest(k).coord,
q_new.coord)*ParamCost+Costeang*(1-ParamCost);
%               line([q_min.coord(1), q_new.coord(1)],
[q_min.coord(2), q_new.coord(2)], 'Color', 'g','LineWidth', 0.1);
               actualizar=true;
               k=posicion;
               hold on
           end

         else

            if
noCollision2(q_nearest(k).coord,q_goal(aux).coord,q_goal(aux).theta,th
reshold)
               if
noCollision2(q_new.coord,q_goal(aux).coord,q_goal(aux).theta,threshold
) && (q_nearest(k).cost + dist(q_nearest(k).coord,
q_new.coord)*ParamCost+Costeang*(1-ParamCost)) <= C_min
               q_min = q_nearest(k);
               C_min = q_nearest(k).cost + dist(q_nearest(k).coord,
q_new.coord)*ParamCost+Costeang*(1-ParamCost);
 %               line([q_min.coord(1), q_new.coord(1)],
[q_min.coord(2), q_new.coord(2)], 'Color', 'g','LineWidth', 0.1);
               actualizar=true;
               k=posicion;
               hold on
               end

            elseif noCollision2(q_nearest(k).coord,q_goal(aux-
1).coord,q_goal(aux-1).theta,threshold)

               if
noCollision2(q_new.coord,q_goal(aux).coord,q_goal(aux).theta,threshold
) && noCollision2(q_new.coord,q_goal(aux-1).coord,q_goal(aux-
1).theta,threshold) && (q_nearest(k).cost + dist(q_nearest(k).coord,
q_new.coord)*ParamCost+Costeang*(1-ParamCost)) <= C_min
               q_min = q_nearest(k);
               C_min = q_nearest(k).cost + dist(q_nearest(k).coord,
q_new.coord)*ParamCost+Costeang*(1-ParamCost);
%               line([q_min.coord(1), q_new.coord(1)],
[q_min.coord(2), q_new.coord(2)], 'Color', 'g','LineWidth', 0.1);
               actualizar=true;
               k=posicion;
               hold on
```

```
                end


              end

          end

        end

        % Se actualiza padre y coste del nodo más cercano
        for j = 1:1:length(nodes)
            if nodes(j).coord == q_min.coord
                q_new.parent = j;
                if actualizar==true

                    if (q_nearest(posicion).parent==0) && (aux==1)
                        Costeang=0;
                    elseif (q_nearest(posicion).parent==0) && (aux>1)
                       Angulo1=angle(q_new.coord(2)-
q_nearest(posicion).coord(2),q_new.coord(1)-
q_nearest(posicion).coord(1));
                       Angulo2=Angulo_Llegada;
                       Costeang=(abs(Angulo1-Angulo2))^2;
                    else
                        Angulo1=angle(q_new.coord(2)-
q_nearest(posicion).coord(2),q_new.coord(1)-
q_nearest(posicion).coord(1));
                        Angulo2=angle(q_nearest(posicion).coord(2)-
nodes(q_nearest(posicion).parent).coord(2),q_nearest(posicion).coord(1
)-nodes(q_nearest(posicion).parent).coord(1));
                        Costeang=(abs(Angulo1-Angulo2))^2;
                    end

                    if Costeang>32400

                     if Angulo1>Angulo2
                        Costeang=(360+Angulo2-Angulo1)^2;
                     else
                        Costeang=(360+Angulo1-Angulo2)^2;
                     end

                    end


                  q_new.cost=q_nearest(posicion).cost +
dist(q_nearest(posicion).coord, q_new.coord)*ParamCost+Costeang*(1-
ParamCost);
                end
            end
        end

        % Append to nodes
        nodes = [nodes q_new];
     end


end
```

```
threshold=threshold2;

D = [];
for j = 1:1:length(nodes)
    tmpdist = dist(nodes(j).coord, q_goal(aux).coord);
    D = [D tmpdist];
end

% Search backwards from goal to start to find the optimal least cost
path
[val, idx] = min(D);
%q_final = nodes(idx);

node_goal(aux).parent=padre_goal;
%q_goal.parent = padre_goal;
q_end = node_goal(aux);
nodes = [nodes node_goal(aux)];
padres=[padre_goal];
while q_end.parent ~= 0
    start = q_end.parent;
    figure(2)
    line([q_end.coord(1), nodes(start).coord(1)], [q_end.coord(2),
nodes(start).coord(2)], 'Color', 'r', 'LineWidth', 1);
    drawnow
    hold on
    figure(1)
    line([q_end.coord(1), nodes(start).coord(1)], [q_end.coord(2),
nodes(start).coord(2)], 'Color', 'r', 'LineWidth', 1);
    drawnow
    hold on
    q_end = nodes(start);
    padres=[padres q_end.parent];
end

%clear nodes;
new_umbral=false;
Angulo_Llegada=angle(node_goal(aux).coord(2)-
nodes(padre_goal).coord(2),node_goal(aux).coord(1)-
nodes(padre_goal).coord(1));
final_node=length(nodes);

padres=[final_node padres];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%5%%%%%CONTROLADOR%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

num_fathers=length(padres)-2;
    if aux==1
        a=q_start.coord(1);
        b=q_start.coord(2);
        ang1=angle(nodes(padres(num_fathers)).coord(2)-
q_start.coord(2),nodes(padres(num_fathers)).coord(1)-
q_start.coord(1));
    else
        ang1=angle(nodes(padres(num_fathers)).coord(2)-
nodes(1).coord(2),nodes(padres(num_fathers)).coord(1)-
nodes(1).coord(1));
    end

    ang=ANG_to_Ctrl(ang1);
```

```
x=a;
y=b;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%5%%%%%GRAFICADO%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%

while num_fathers>0

    if aux==1 && num_fathers==length(padres)-2

xe=Errorx(q_start.coord(1),nodes(padres(num_fathers)).coord(1),q_start
.coord(2),nodes(padres(num_fathers)).coord(2),x,y);
    else

xe=Errorx(nodes(padres(num_fathers+1)).coord(1),nodes(padres(num_fathe
rs)).coord(1),nodes(padres(num_fathers+1)).coord(2),nodes(padres(num_f
athers)).coord(2),x,y);
    end


    %Equaciones

    x=x-V*sin(Ttheta)*hh;
    y=y+V*cos(Ttheta)*hh;
    Ttheta=Ttheta+V*gamma*hh;

    a=[a x];
    b=[b y];

    if (Ttheta>0) && (ang<-pi/2)
       ref_errorang=-Ttheta-ang;
    else
       ref_errorang=Ttheta-ang;
    end
    % referencia=evalfis([xe,ref_errorang],pract2);
   if ang1<=180 && ang1>=0
    referencia=evalfis([xe,ref_errorang],pract2);
   else
    referencia=evalfis([-xe,ref_errorang],pract2);
   end

    gamma=gamma+((1/T)*(referencia-gamma)*hh);


    if x+0.1>nodes(padres(num_fathers)).coord(1) & x-
0.1<nodes(padres(num_fathers)).coord(1)
        if y+1>nodes(padres(num_fathers)).coord(2) & y-
1<nodes(padres(num_fathers)).coord(2)
            if num_fathers>1
                num_fathers=num_fathers-1;
 %               x1=nodes(padres(num_fathers)).coord(1);
                ang1=angle(nodes(padres(num_fathers)).coord(2)-
nodes(padres(num_fathers+1)).coord(2),nodes(padres(num_fathers)).coord
(1)-nodes(padres(num_fathers+1)).coord(1));
                ang=ANG_to_Ctrl(ang1);
            else
                num_fathers=num_fathers-1;
            end
```

```
        end
    end
end

      figure(2)
      plot(a,b,'b')

a=a(length(a));
b=b(length(b));
clear nodes;


end
```

2- FUNCIONES

Crecimiento

```
function A = steer(qr, qn, val, eps)
   qnew = [0 0];

   % Se mueve hasta qn con una distancia máxima definida por eps
   if val >= eps
       qnew(1) = qn(1) + ((qr(1)-qn(1))*eps)/val;
       qnew(2) = qn(2) + ((qr(2)-qn(2))*eps)/val;
   else
       qnew(1) = qr(1);
       qnew(2) = qr(2);
   end
   A = [qnew(1), qnew(2)];
end
```

Detección carretera

```
function nc = noCollision2(newnode,goal,angulo,umbral)

   var1=goal(2)-goal(1)*tand(angulo);

   if tand(angulo)==Inf || tand(angulo)==-Inf

        if (newnode(1)<=goal(1)+umbral && newnode(1)>=goal(1)-umbral)
            nc=1;
        else
           nc=0;
        end

   elseif tand(angulo)==0

      street1=@(x) var1+umbral;
      street2=@(x) var1-umbral;

    limit1=street1(newnode(1));
    limit2=street2(newnode(1));

        if (newnode(2)<=limit1 && newnode(2)>=limit2)
```

```
            nc=1;
        else
           nc=0;
        end

   else

   %graficado de la recta
   street1=@(x) tand(angulo)*x+var1+umbral/cosd(angulo);
   street2=@(x) tand(angulo)*x+var1-umbral/cosd(angulo);

    limit1=street1(newnode(1));
    limit2=street2(newnode(1));

    if (angulo<90) || (angulo>270)
        if (newnode(2)<=limit1 && newnode(2)>=limit2)
            nc=1;
        else
           nc=0;
        end

    else

        if (newnode(2)<=limit2 && newnode(2)>=limit1)
            nc=1;
        else
           nc=0;
        end
    end

   end

 end
```

Detección obstáculos

```
function nc = noCollision(n2, n1, o)
    A = [n1(1) n1(2)];
    B = [n2(1) n2(2)];
    obs = [o(1) o(2) o(1)+o(3) o(2)+o(4)];

    C1 = [obs(1),obs(2)];
    D1 = [obs(1),obs(4)];
    C2 = [obs(1),obs(2)];
    D2 = [obs(3),obs(2)];
    C3 = [obs(3),obs(4)];
    D3 = [obs(3),obs(2)];
    C4 = [obs(3),obs(4)];
    D4 = [obs(1),obs(4)];

    % Comprueba que entre q_near y q_rand no se interpone el objeto
    % Check if path from n1 to n2 intersects any of the four edges of
the
    % obstacle

    ints1 = ccw(A,C1,D1) ~= ccw(B,C1,D1) && ccw(A,B,C1) ~=
ccw(A,B,D1);
```

```
    ints2 = ccw(A,C2,D2) ~= ccw(B,C2,D2) && ccw(A,B,C2) ~=
ccw(A,B,D2);
    ints3 = ccw(A,C3,D3) ~= ccw(B,C3,D3) && ccw(A,B,C3) ~=
ccw(A,B,D3);
    ints4 = ccw(A,C4,D4) ~= ccw(B,C4,D4) && ccw(A,B,C4) ~=
ccw(A,B,D4);
    if ints1==0 && ints2==0 && ints3==0 && ints4==0
        nc = 1;
    else
        nc = 0;
    end
end
```

Cálculo de intersecciones

```
function val = ccw(A,B,C)
    val = (C(2)-A(2)) * (B(1)-A(1)) > (B(2)-A(2)) * (C(1)-A(1));
end
```

Cálculo de la distancia entre nodos

```
function d = dist(q1,q2)
      d = sqrt((q1(1)-q2(1))^2 + (q1(2)-q2(2))^2);
end
```

Cálculo del ángulo

```
function ang = angle(numerador,denominador)

if numerador <=0 && denominador<0

  ang=atand(numerador/denominador)+180;

elseif  numerador >0 && denominador<0

  ang=atand(numerador/denominador)+180;

elseif  numerador >=0 && denominador>0

  ang=atand(numerador/denominador);

elseif  numerador <0 && denominador>0

  ang=atand(numerador/denominador)+360;

elseif denominador==0 && numerador<0
  ang=270;

elseif denominador==0 && numerador>0
  ang=90;
elseif denominador==0 && numerador==0
   ang=0;
end
end
```

Cálculo del ángulo para la correcta lectura del controlador

```
function RAD = ANG_to_Ctrl(DEGR)

RAD=DEGR*(pi/180);

if DEGR<=270 && DEGR>=0
    RAD=RAD-pi/2;
else
    RAD=-(RAD-(pi-(2*pi-RAD)*2))+pi/2;

end

end
```

Cálculo del error en x

```
function Error = Errorx(x1,x2,y1,y2,xa,ya)

       Recta1=@(x) ((y2-y1)*x)/(x2-x1)-((y2-y1)*x1)/(x2-x1)+y1;
      % pendiente=(y2-y1)/(x2-x1);

      %Recta1=@(x) ((-y2*x1+y1*x1+y1*x2-y1*x1)+x*(y2-y1))/(x2-x1);
      pendiente= (y2-y1)/(x2-x1);

      Recta2=@(x) (-pendiente^-1)*(x-xa)+ya;

      SetX=-(((pendiente^-1)*xa)+ya+((y2-y1)*x1)/(x2-x1)+y1)/(-
      pendiente^-1-pendiente);

      %SetX= (ya+((y2-y1)*x1)/(x2-x1)-y1)/pendiente;

      SetX = (xa+pendiente*ya-((-y2*x1+y1*x1+y1*x2-
      y1*x1)*pendiente)/(x2-x1))/(pendiente^2+1);

      %Error=sqrt((xa-SetX)^2+(Recta1(SetX)-ya)^2);
      Error=xa-SetX;

end
```

Graficado de las restricciones en las carreteras

```
function street = carretera(goal,angulo,umbral)

   var1=goal(2)-goal(1)*tand(angulo);

   if (tand(angulo)==Inf) || (tand(angulo)==-Inf)

        line([goal(1)+umbral goal(1)+umbral], get(gca, 'ylim'))
        hold on
        line([goal(1)-umbral goal(1)-umbral],get(gca, 'ylim'))
        plot(goal(1),goal(2),'r*')
        hold on

   elseif tand(angulo)==0
```

```
        street1=@(x) var1+umbral;
        street2=@(x) var1-umbral;
        fplot(street1,[-1000 1000])
        hold on
        fplot(street2,[-1000 1000])
        plot(goal(1),goal(2),'r*')
        hold on

    else

        %graficado de la recta
        street1=@(x) tand(angulo)*x+var1+umbral/cosd(angulo);
        street2=@(x) tand(angulo)*x+var1-umbral/cosd(angulo);

        fplot(street1,[-1000 1000])
        fplot(street2,[-1000 1000])
        plot(goal(1),goal(2),'r*')
        drawnow
        hold on
    end
street=1;

end
```

Circuito 1

```
y1=0:1:350;
x1=300+200*sin((0.2*pi/20)*y1);

y2=0:1:350;
x2=500+200*sin((0.2*pi/20)*y2);

coordx  = [550 400 220 550 400 ];
coordy  = [50 100 150 250 300];

figure(2)
for i=1:length(coordx)
    plot(coordx(i),coordy(i),'r*')
    hold on
end

obstacle = [520,15,20,20];
obstacle1= [315,100,40,20];
obstacle2= [450,200, 10,23];
rectangle('Position',obstacle,'FaceColor',[0 .5 .5])
rectangle('Position',obstacle1,'FaceColor',[0 .5 .5])
rectangle('Position',obstacle2,'FaceColor',[0 .5 .5])
hold on

plot(x1,y1,'black')
xlim([-1 700])
hold on
plot(x2,y2,'black')
xlim([-1 700])
hold on
```

Circuito 2

```
figure(2)

coordx  = [50 50 300 400 450 450 250];
coordy  = [50 400 450 450 400 200 300 ];

for i=1:length(coordx)
    plot(coordx(i),coordy(i),'r*')
    hold on
end


line([0, 0], [0, 500], 'Color', 'k', 'LineWidth', 1);
line([100, 100], [0, 400], 'Color', 'k', 'LineWidth', 1);

line([0, 500], [500, 500], 'Color', 'k', 'LineWidth', 1);
line([100, 400], [400, 400], 'Color', 'k', 'LineWidth', 1);

line([400, 400], [400, 250], 'Color', 'k', 'LineWidth', 1);
line([500, 500], [500, 150], 'Color', 'k', 'LineWidth', 1);

line([400, 300], [250, 250], 'Color', 'k', 'LineWidth', 1);
line([500, 200], [150, 150], 'Color', 'k', 'LineWidth', 1);

line([200, 200], [150, 350], 'Color', 'k', 'LineWidth', 1);
line([300, 300], [250, 350], 'Color', 'k', 'LineWidth', 1);
line([200, 300], [350, 350], 'Color', 'k', 'LineWidth', 1);
```

Circuito 3

```
figure(2)

coordx  = [200 600 750 ];
coordy  = [45 220 600 ];

for i=1:length(coordx)
    plot(coordx(i),coordy(i),'r*')
    hold on
end

line([0, 390], [70, 70], 'Color', 'k', 'LineWidth', 1);
line([0, 410], [20, 20], 'Color', 'k', 'LineWidth', 1);

line([390, 710], [70, 350], 'Color', 'k', 'LineWidth', 1);
line([410, 780], [20, 350], 'Color', 'k', 'LineWidth', 1);

line([710, 710], [350, 800], 'Color', 'k', 'LineWidth', 1);
line([780, 780], [350, 800], 'Color', 'k', 'LineWidth', 1);

obstacle = [50,20,100,40];
obstacle1= [300,30,100,40];
obstacle2= [710,350, 60,200];
rectangle('Position',obstacle,'FaceColor',[0 .5 .5])
rectangle('Position',obstacle1,'FaceColor',[0 .5 .5])
rectangle('Position',obstacle2,'FaceColor',[0 .5 .5])
hold on
```